Spring 5-24-2017

Community Detection in Social Networks

Ketki Kulkarni
San Jose State University

Community Detection in Social Networks

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Ketki Kulkarni

May 2017

The Designated Project Committee Approves the Project Titled

Community Detection in Social Networks

by

Ketki Kulkarni

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2017

Katerina Potika	Department of Computer Science
Sami Khuri	Department of Computer Science
Margareta Ackerman	Department of Computer Science

ABSTRACT

Community Detection in Social Networks

by Ketki Kulkarni

The rise of the Internet has brought people closer. The number of interactions between people across the globe has gone substantially up due to social awareness, the advancements of the technology, and digital interaction. Social networking sites have built societies, communities virtually. Often these societies are displayed as a network of nodes depicting people and edges depicting relationships, links. This is a good and efficient way to store, model and represent systems which have a complex and rich information. Towards that goal we need to find effective, quick methods to analyze social networks. One of the possible solution is community detection. The community detection deals with finding clusters, groups in a network. Detecting such communities is very important in many fields in order to understand and extract the information from complex systems. The problem is very hard and has been studied extensively for the past few years. With this project, we will define the problem, study existing methods, propose new methods, and experimentally evaluate them using synthetic and real datasets. Additionally, we will describe applications to smart city communities and challenges that have to be resolved.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Prof. Katerina Potika, who expertly guided me through my graduate education and my master's project. Her unwavering enthusiasm for the study of social networks kept me engaged with my research. Her constant mentorship, advice and support helped me to move in a right direction towards completion of the project. I would like to thank her for her time, help and efforts towards me and this project.

My deep gratitude also goes to Prof. Sami Khuri and Prof. Margareta Ackerman for being on my defense committee. I would like to thank them for their time and efforts.

Lastly, I would like to thank my friends and family. They supported and helped me to survive this stress and not letting me give up.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Over the last few decades, advancement in the technology, the rise of the Internet have led to virtual groups, communities and societies where people interact, and share information. Such a network of interactions and personal relationships is called a Social Network. Many such networking sites like Facebook, Twitter, LinkedIn, GitHub, Instagram et al. provide many services to the user like providing a platform to share opinions, meet new people, stay in touch with old friends etc. All these sites have the same objective i.e. to let people communicate effectively and efficiently.

The study and analysis of these virtual networks is becoming a popular topic among the researchers. These networks are so rich with the information that the scientist from different disciplines like sociology, computer science, anthropology, psychology et al. are focusing on examining different structural and statistical properties of social networks in order to leverage them to improve their everyday operations [10]. Predicting human emotions, sentiment analysis are few applications of these researches.

One of the problem of analyzing these networks is the amount of information that need to be stored and retrieved efficiently. Many data structures provide excellent methods for storage, but there is still need of cost-effective and productive methods for data retrieval. One such method is the community detection.

Community in a large network is a collection of nodes. The nodes within communities are more densely connected to each other than that of belonging to different communities. This helps us to find the nodes which have common properties and

evaluate relationships between them. For example, if we are to build a network of the users of Facebook, then nodes will depict the users and the links will represent the relationships between the users and the community will be a group of users who follow let's say the same football club or support the same presidential nominee. Finding such groups allows us to evaluate individual users, the interaction between these users and predict the missing information. Therefore, analyzing the information in the real-world networks and detecting communities have become a very important problem in various areas.

Communities can be implicit or explicit. Communities that are not actually built by its group members but formed by a third party comes under implicit category. For example, Yahoo! groups come under explicit community, whereas a community in which all the people who use similar or same programming languages come under implicit. In most of the social networking sites, contrast to explicit communities, implicit communities and their members are obscure to many people [28].

The problem of community detection revolves around finding such implicit communities in which the nodes that exhibit similar properties or behaviors are grouped together. Currently, there are several methods and techniques that deal with finding community structure. As an example a lot of technologies identify edges that connect different groups. To find such an edge various centrality measures, further explained in Section 2.1, are used. An alternative approach is to find a hierarchical structure in a network. Another approach is to categorize known nodes into groups in order to maximize/minimize some cost function. Some of the algorithms based on these approaches are Girvan-Newman algorithm based on edge betweenness [11], Louvain algorithm [2], Label propagation algorithm [23] etc.

1.1 Problem Definition

After studying and analyzing the existing community detection algorithms, we are proposing three algorithms. These algorithms are the combination of both, top-down and bottom-up, Hierarchical Clustering methods (Section 2.1) and (Section 2.1). The top-down approach breaks the large chunk of vertices, grouped together, into smaller chunks. Whereas, the bottom-up approach merges the smaller partitions into bigger ones. In this project, we are proposing two approaches for the bottom-up method and one approach for the top-down method.

Often, edges within same communities tend to have smaller edge betweenness ratio as compared to that of edges belonging to different communities. Different centrality measures will help us to decide the importance of an edge in a network. In our bottom-up strategy, we plan to add edges to the output graph based on these centrality measure i.e. iteratively merge the network. And in the top-down strategy we plan to remove the edges from the output graph i.e. iteratively break down the network.

While breaking or merging the network into partitions, we plan to keep track of the quality and the quantity of these formed partitions using a measure called the Modularity Q (chapter 2).

After achieving this, we are aiming to compare and contrast the new proposed algorithms with the existing algorithms. We are also planning to analyze the effects of this algorithm on the random graphs and real-world networks.

CHAPTER 2

Terminology

2.1 Graph Terminology

A graph is a set of vertices connected by edges as shown in Fig. 1. Often a graph is denoted by $G = (V, E)$ where V and E are lists of nodes and edges respectively. In the following example, V is a set of nodes $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)$ and E is a group of edges connecting nodes like $(1-7, 6-7, 3-5, 3-4, ..., 2-6)$.

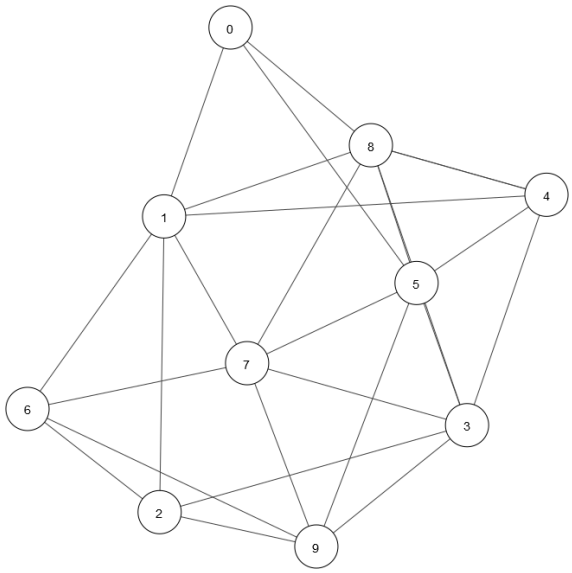


Figure 1: Example of simple graph with 10 nodes labeled as [0-9] connected by edges.

Adjacency List and Adjacency Matrix: A graph can be written as an adjacency list and an adjacency matrix. For a graph $G = (V, E)$, an adjacency matrix A is represented by a matrix of size $n * n$, where n = the number of nodes $|V|$. For a weighted graph, $A_{uv} = 1$ if u and v are connected by an edge, otherwise 0. For

an unweighted graph, $A_{uv} = w$ where w is the weight of the edge joining nodes u and v. An adjacency matrix of a graph shown in Fig.1 is given in Fig.2a. For both un-directed and directed graphs, $A_{uv} = 0$ if $u = v$

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	0	1	0	0	1	0
1	1	0	1	0	1	0	1	1	1	0
2	0	1	0	1	0	0	1	0	0	1
3	0	0	1	0	1	1	0	1	0	1
4	0	1	0	1	0	1	0	0	1	0
5	1	0	0	1	1	0	0	1	1	1
6	0	1	1	0	0	0	0	1	0	1
7	0	1	0	1	0	1	1	0	1	1
8	1	1	0	0	1	1	0	1	0	0
9	0	0	1	1	0	1	1	1	0	0

(a) Adjacency matrix

0	1 → 5 → 8
1	0 → 2 → 4 → 6 → 7 → 8
2	1 → 3 → 6 → 9
3	2 → 4 → 5 → 7 → 9
4	1 → 3 → 5 → 8
5	0 → 3 → 4 → 7 → 8 → 9
6	1 → 2 → 7 → 9
7	1 → 3 → 5 → 6 → 8 → 9
8	0 → 1 → 4 → 5 → 7
9	2 → 3 → 5 → 6 → 7

(b) Adjacency list

Figure 2: An adjacency matrix and adjacency list

An adjacency list is represented as an indexed array of list. Each node has an un-ordered list of nodes which are neighbors of that node. Each edge is represented twice, one for source node and another for destination. Example of an adjacency list is given in Fig.2b.

Bridge and Local Bridge: Another important terms are a bridge and a local bridge. [8] defines a bridge as the link between node A and B if deletion of that edge divides node A and node B into two different groups i.e. if that edge was the only

connection between A and B. Whereas local bridge is defined as an edge between A and B if the deletion of that edge would extend the path between A and B.

Centrality: All the community detection algorithms are based on one or more centrality measures. Centrality measure defines how important the node is in a given network [28]. It also defines how central a node is to its community. Following are some of the centralities:

1. Edge/vertex betweenness
2. Degree
3. Closeness
4. Page Rank
5. Katz
6. Eigenvector
7. Cross-clique

Degree centrality: People who have lots of contacts and connections are usually considered important [28]. We can relate this to graph theory. The degree centrality measures the rank of node based on its connections. A degree is the number of the edges connected to a node. Degree of node v_i is often represented as d_i [28]. Thus, it can be defined as the number of the neighbors of the given node. The degree centrality C_d of a node v_i in un-directed graph is calculated as

$$C_d(v_i) = d_i \tag{1}$$

where d_i = the degree of the node v_i. It calculates how popular or how well connected a node is in a network.

Edge Betweenness: Out of all these centralities, we are going to focus on edge betweenness (EB). For a graph $G = (V, E)$, edge betweenness of an edge $v \in V$ defines how important that edge is. It specifies how much network traffic passes through that edge. Basically it counts the shortest paths between two nodes, say i and j, passing through node k. It's based on the assumption that if most of the flow of network pass through an edge then that edge is important and how much it is important is measured by betweenness centrality. It is calculated using formula,

$$C_B(n_i) = \sum_{j<k} \frac{g_{jk}(n_i)}{g_{jk}} \tag{2}$$

where g_{jk} = the count of the shortest paths joining j and k, $g_{jk}(n_i)$ = the number that actor i is on.

Neighborhood Overlap: The neighborhood overlap (NOVER) of an edge (u, v) is the ratio of the number of common neighbors of both u and v to the number of nodes that are neighbors of either u or v. It is an embeddedness divided by total number of neighbors of both nodes connected by that edge.

$$NOVER(u, v) = \frac{\|N_u \cap N_v\|}{\|N_u \cup N_v\| - 2} \tag{3}$$

When the edge between two nodes is a local bridge then $NOVER = 0$. So the notion of a local bridge is contained within this definition and hence we can think of the edges with very small NOVER value as being almost local bridges.

Modularity: The quality of the community is measured using a modularity principle. Girvan and Newman proposed a new function [21] which evaluates community structure. Modularity Q is a scalar value such that $-1 \leq Q \leq 1$, and it measures

the density of the vertices within the same community to that of nodes belonging to a different community. The larger the modularity score, the more appropriate is the partitioning of the nodes into communities. It is used to compare the communities obtained by different algorithms/methods. It is calculated as,

$$Q = \frac{1}{2m} \cdot \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \cdot \delta(c_i, c_j) \tag{4}$$

Here m = the number of edges, A_{ij} = the weight of an edge between nodes i and j, k_i = a degree of node i, c_i = the community to which i belongs to and δ = a function such that $\delta(u, v) = 1$ if $u = v$ else 0. As per the fast greedy algorithm [4], it is a property of a network and specific proposed division of that network into communities .

Different methods of Graph Partition: Now we will focus on the different graph partitioning methods. In [10], they have specified 4 graph clustering methods in

Hierarchical clustering: Many social networks have hierarchical structures i.e. clusters within clusters and so on. Hierarchical Clustering aims to find such multilevel structures in social networks [10]. The starting point for any hierarchical clustering algorithm is to find the similarities and differences between vertices. This helps in finding the cluster of vertices that have similar features i.e. based on the vertex similarity. Usually, this approach is represented by a dendrogram like shown in Fig.3 This method is further classifies as,

1. Divisive: A top-down approach
2. Agglomerative: A bottom-up approach

Unlike the graph partitioning, the hierarchical clustering does not require a prior

knowledge of the number or the size of the communities within the network. This is one of the advantages of this method. One of the disadvantage of this approach is that vertices are not correctly classified or vertices with a degree=1 are classified as a separate cluster.

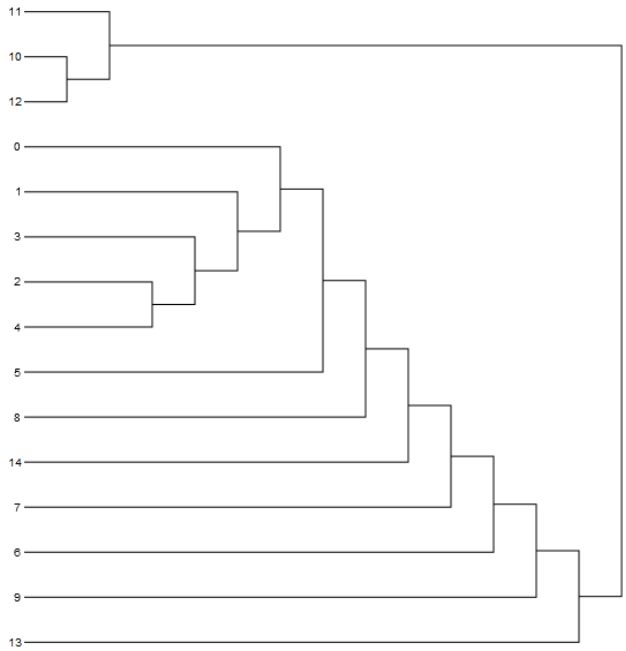


Figure 3: Example of the dendrogram

Divisive Hierarchical Clustering: This is a top-down approach in which a large network is iteratively split into different chunks based on different centrality measures [28]. Some of the algorithms using this approach are Girvan-Newman algorithm [11], Neighborhood overlap (NOVER) based community detection [17] etc.

Agglomerative Hierarchical Clustering This is a bottom-up approach in which iteratively small communities are merged into a larger community to maximize some cost function. One of the disadvantage of this approach is that it does not scale well. Some of the algorithms implementing this approach are Louvain algorithm [2], Label propagation algorithm [23] etc.

Graph partition: This approach divides the network into partitions [10]. The nodes are are divided into partitions, also called cuts, of fixed size. The partitions are formed as long as the number of the edges between them are less. This is a NP-hard problem. Example of this approach is given in Fig.4. [13] proposed an algorithm. It uses an evaluation function of the difference between the intra and inter community links. During local search process, this algorithm only accepts best solutions and hence gives mostly local optimal solutions.

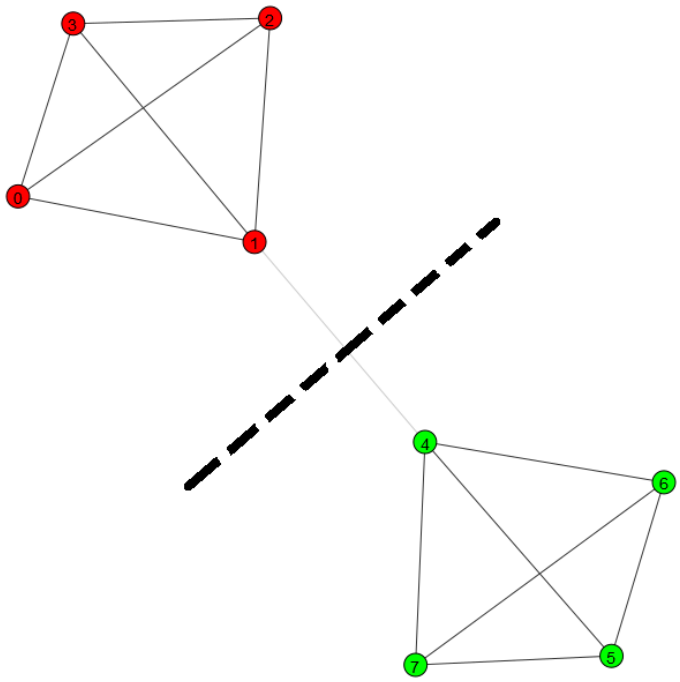


Figure 4: Example of the Graph Partition method.

Spectral clustering: Given a set of n objects $x_1, x_2, x_3, ..., x_n$ and a similarity function S such that

$$S(x_i, x_j) = S(x_j, x_i) \geq 0, \forall i, j = 1, 2, ..., n \tag{5}$$

spectral clustering includes all the methods that group the nodes using eigenvectors of matrices like S [10]. It consist of a transformation of the given set of points

into set of points in space. The co-ordinates of these points are the elements of the eigenvectors. Once the transformation is complete, the points are clustered using standard methods like k-mean clustering [15]. Most of the algorithms based on this method use a Laplacian matrix. One of the algorithms that uses this approach is a community detection using random walks by Pons et al. [22].

Partitional clustering In this approach, the number of clusters or communities is predefined, say k, such that the nodes are partitioned into k clusters [10]. The aim is to maximize or minimize certain cost function. Most of the time the distance between nodes is used to group them together. Some of the popular cost functions are,

1. Minimum k-clustering: Popular data analysis algorithm. It partitions n points into k clusters in which each point belongs to the cluster with the nearest mean. The cost function is the diameter of the cluster which is the largest distance between two nodes of a cluster.

2. k-clustering sum: Similar to k-mean clustering. The cost function is the average distance between all pairs of points of a cluster.

3. k-center: For each cluster i a centroid x_i is calculated and the maximum d_i of the distances of each node is calculated from centroid.

4. k-median: Similar to k-center function. But the cost function is the average distance from the centroid.

The most popular technique is k-mean clustering [15].

Connected Component: An undirected graph is connected if every vertex is reachable from every other vertex [5]. It is a subgraph of a graph where every vertex

is reachable from all other vertices in a subgraph. Example of subgraph is given in Fig.5, where there are 3 connected components:{7, 8}, {4, 5, 6}, {0, 1, 2, 3}. A undirected graph is called connected if it has exactly one connected component[5].

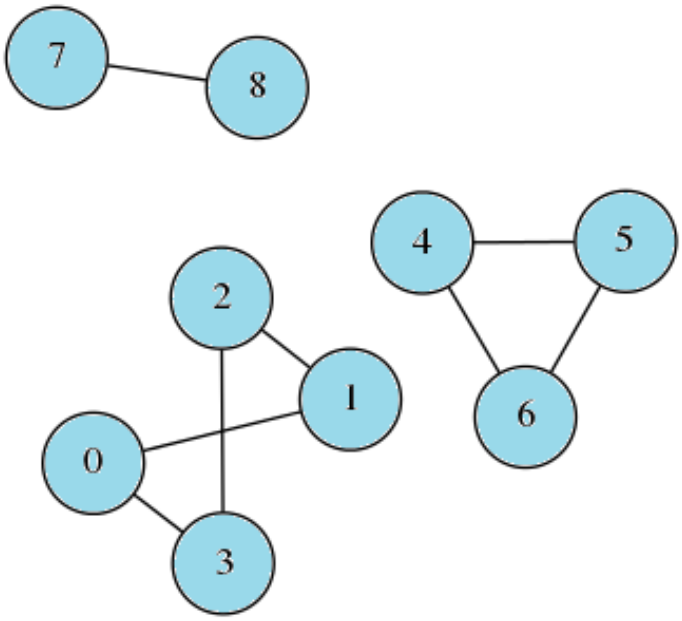


Figure 5: Example of Connected Component

Breadth-First Search Algorithm: The Breadth-First Search (BFS) is a traversing algorithm for trees and graphs. It starts with a root node and traverses through neighbors of root vertex first and then neighbors of neighbors of root and so on. It uses queue data structure to keep track of which vertex to visit next. It visits every vertex only once. It takes a graph $G = (V, E)$ and vertex *root* as input and finds all the vertices traversable by *root* vertex. This algorithm runs in $\mathcal{O}(|V| + |E|)$ time. Its algorithm given by Algorithm 1.

Minimum Spanning Tree (MST): Given a connected graph $G = (V, E)$ with real-valued edge weights c_e, a MST is a subset of edges $T \subseteq E$ such that T is a

Algorithm 1 Breadth-First Search

Input: A graph $G = (V, E)$ and a vertex $v \in V$
Output: A subset of vertices $T \subseteq V$ reachable from vertex v

```
1:  Procedure BFS(G, v)
2:      T ← φ
3:      Queue Q ← φ
4:      Q.enqueue(v)
5:      while Q is not empty do
6:          v' ← Q.dequeue()
7:          for each neighbor n of v' do
8:              if n is not visited then
9:                  T ← T ∪ {n}
10:                 Q.enqueue(n)
11:             end if
12:         end for
13:     end while
14: end Procedure
15: return T
```

spanning tree for which the sum of edge weights is minimum [25]. Example of MST is shown in Fig.6 and Fig.7. MST has many diverse applications such as network design, cluster analysis, image processing et. al. One of the famous algorithm which produces an MST is Kruskal's algorithm [14].

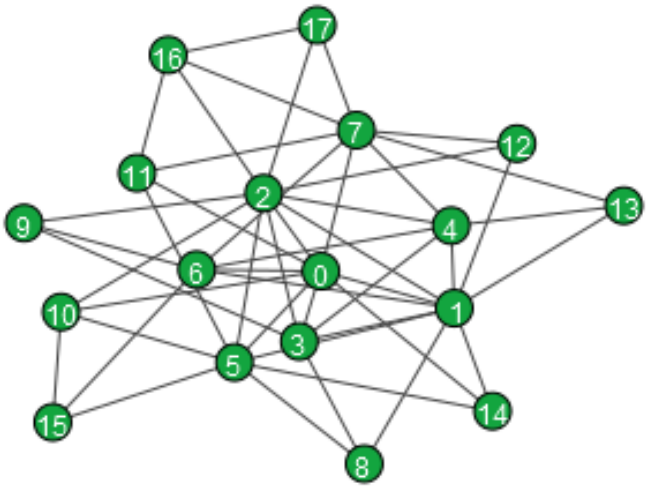


Figure 6: A graph with 18 vertices and 48 edges

Kruskal's algorithm: It is a greedy approach to find minimum weighted edge which combined two trees into forest. It connects two trees unless joining them would create a cycle. This algorithm makes use of Union-Find data structure and it maintains the set of connected component. Given a graph $G = (V, E)$, this algorithm builds a MST in $\mathcal{O}(E \log V)$ time. Its algorithm given by Algorithm 2.

Algorithm 2 Kruskal's algorithm

Input: A weighted graph $G = (V, E)$
Output: A set T of edges in the MST such that $T \subseteq E$

```
1: Procedure KRUSKAL(G = (V, E))
2:     Sort the edges by their weights
3:     T ← φ
4:     for each (u ∈ V) do
5:         Make a set containing singleton u
6:     end for
7:     for i = 1 to m do
8:         (u, v) ← e_i
9:         if u and v are in different connected component set then
10:            T ← T ∪ {e_i}
11:            Merge the sets containing u and v
12:        end if
13:    end for
14:    return T
15: end Procedure
```

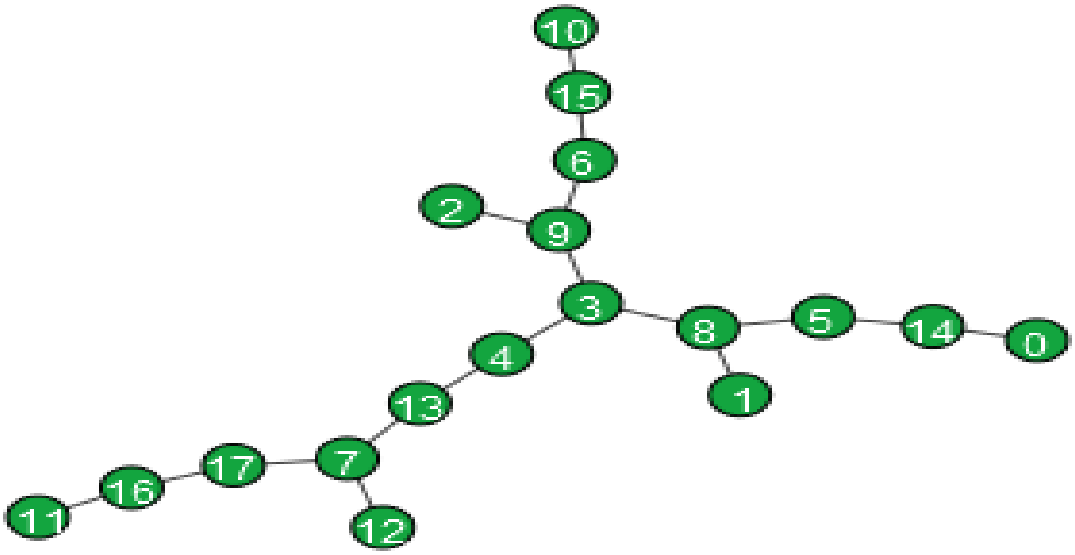


Figure 7: Example of MST using Kruskal's algorithm

In the next chapter, we will cover existing methodology and algorithms for community detection.

CHAPTER 3

Existing methods for community detection

In recent years, researchers are more focusing on structural and statistical properties of social networks. The power-law degree distribution, small-world phenomena, and network transitivity are used to analyze the social networks as these properties are most common in all networks. Another such property is formed communities. Community is a set of nodes such that ratio of links within communities is larger as compared to that of between communities.

Girvan, Newman et al. [11] start with explaining traditional methods of community detection, then explaining their drawbacks and finally explaining their proposed algorithm. The traditional method mentioned in this paper is a hierarchical clustering approach. This process starts off by calculating weights W_{ij} for every nodes i and j. Nodes are sorted according to the weights and then two nodes are linked in order of their weights. As each two nodes are linked hierarchical structure of community is explored. There are two ways mentioned to define weights,

1. the node-independent track between nodes. Two paths are node-independent if they have same end-points but do not have any other common points.

2. all the paths between nodes.

Both of these approaches have drawbacks. If a vertex is connected to the rest of the network by only one edge, then it should be in the community of the other end of an edge. But in both of these cases, such single nodes remain isolated while exploring community structure.

To overcome this drawback, Girvan-Newman have proposed an new algorithm in [11]. This algorithm finds most important edges, which are most of the time present between communities. It uses EB as opposed to the traditional vertex betweenness. It first calculates EB values for all the edges and then removes the edges using EB ratio. The removal of the edges changes the shortest paths within network, affects the betweenness of other edges. Hence the betweenness is re-calculated. This entire process continues till every edge in the network is removed.

If a network has several communities connected by few edges, then most of the shortest paths will pass through these few edges and thus making them important. By removing such important edges, communities are identified. Fig.8 shows the example of Girvan-Newman algorithm [11] on the famous Zachary's karate club network [27].

This algorithm was tested for many networks. They generated a synthetic network of 128 nodes. And then partitioned those nodes into 4 communities of 32 nodes each. Let's the probability of edges inside the community be P_{in} and that of between communities be P_{out} such that $P_{out} < P_{in}$. The average degree of each vertex is considered to be 6 i.e. $z_{out} = 6$. This graph was fed to this algorithm which classified 90% of the nodes correctly.

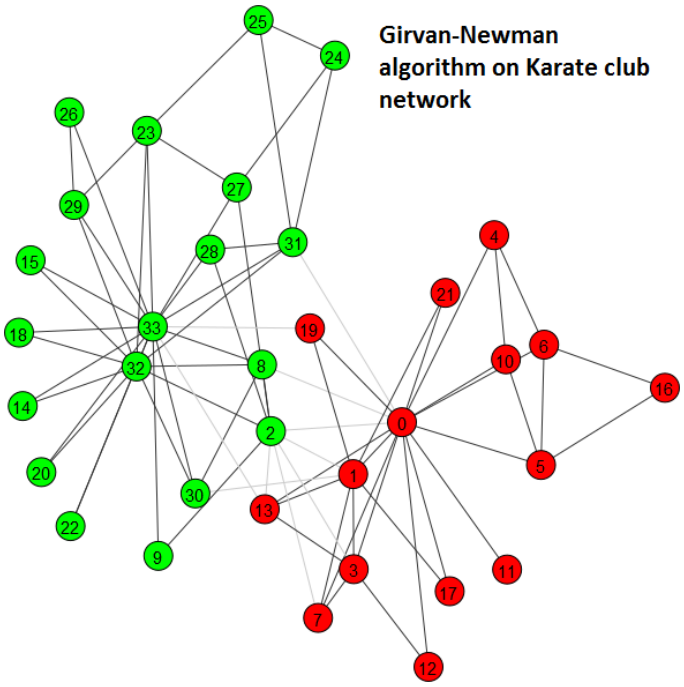


Figure 8: Output of Girvan-Newman algorithm

Another network, on which they have tried this method, is Zachary's karate club [27]. They have used a simple unweighted version of this network. This algorithm efficiently detected the communities with (instructor) 1 and (administrator) 34 being central nodes. Only node 3 was classified incorrectly.

Another is U.S. college football network [19]. They have considered a graphical representation of the schedule of Division I of 2000 season where the nodes are teams and edges represent game between teams [11]. This network has a known community structure in which teams are split into 11 conferences. Each conference contains 8-12 teams. Intra-conference games are more common than inter-conference games. Average number of intra-conference game for each team is 7 whereas that of inter-conference is 4. When applied Girvan-Newman algorithm [11] to this network, it was observed that almost all of the teams were grouped correctly with few exceptions.

Those teams which did not belong to any group were classified to the groups with which they were closely related.

The edge betweenness measure is often used to find clusters. Using this centrality measure either edges are removed or added to the network to find the underlying structure of the network. But these algorithms have tendency to be computationally time consuming and have weakness of not detecting smaller communities correctly. [17] solved this problem by identifying the important edges which if removed from the network will form closely-knit communities with the highest modularity. To identify such important edges Meghanathan has used a measure called NOVER (Equation 2.1).

All edges with NOVER score less than threshold value are weak ties and greater than threshold value are strong ties. All weak ties are removed from the network to form communities. The problem with this approach is deciding the threshold value. Large value of the threshold value could disintegrate the communities or the smaller value could make two different communities as one community.

The algorithm proposed in this paper uses greedy approach to remove edges in increasing order of the NOVER score and thus resulting in the largest modularity score. The NOVER score of edge, which if removed finds the partition with maximum cumulative modularity Q, is called the threshold NOVER. The edges removed until then are weak ties [17].

The idea behind this approach [17] is to use NOVER score to differentiate weak and strong ties. First, the NOVER score of all edges in network is computed and stored in ascending order in a priority queue.

Each iteration removes one edge from the network and the connected components are searched using BFS. Then the modularity of resulting communities and the cumulative modularity of that partition is calculated. The algorithms is continued till all the edges are removed. This algorithm at the end finds the communities which have

the maximum modularity.

The difference between the modularities calculated by this algorithm and that of Girvan-Newman is only 60%. The time complexity of this algorithm is $\mathcal{O}(|E| * (|E| * |V|))$ and execution time of this algorithm is 1% of that of the Girvan-Newman algorithm [11].

It is compared with Girvan-Newman algorithm [11] and Louvain algorithm [2] for following 5 metrics:

1. Cumulative modularity of the best communities detected by each algorithm
2. Execution time in milliseconds
3. Giant community size
4. Resolution limit
5. Normalized mutual information (NMI): the measure of the similarity of communities detected by two different algorithms

This algorithm has introduced a new metric called the (Modularity, Execution time, Resolution, Number of nodes)MERN score to evaluate the community detection algorithms [17]. These divisive hierarchical algorithms break the communities into smaller one. The agglomerative methods do exactly opposite of it i.e. it combines the chunks of the network. One such algorithm is proposed by Newman in [20].

It is an extension of the algorithm proposed in [11]. The main disadvantage of Girvan-Newman algorithm [11] is that it is computationally very slow as it's worst-case time complexity is $\mathcal{O}(m^2n)$ where m and n are the number of nodes and edges respectively. When used on the networks with few thousands nodes, the algorithm fails to compute communities efficiently. To overcome this drawback, Newman proposed a new

algorithm which follows different principles than Girvan-Newman algorithm [11]. This algorithm uses Modularity Q (Equation 2.1). $Q = 0$ specifies that community does not have any more intra-cluster edges than the expected edges and the value greater than 0 indicates randomness and $Q > 0.3$ indicates community structure [4]. These values of Q suggest that instead of removing edges, it should optimize Q and search for best value. Here Newman is considering greedy approach to optimize Q. The process starts with assigning each node with one of the n communities such that number of communities = number of nodes. Two communities are repeatedly merged to maximize the increase or minimize the decrease in Q. It [20] only considers communities between which there are edges because communities without any edges between them never increase the value of Q. The value of Q changes by joining two clusters by,

$$\Delta Q = e_{ij} + e_{ji} - 2a_i a_j = 2(e_{ij} - a_i a_j) \tag{6}$$

Once two communities are merged then corresponding values of e_{ij} are updated, which takes worst-case time $\mathcal{O}(n)$. This algorithm has two advantages:

1. Track of the value Q is kept at each step so it is easy to find the optimal community.

2. Can be used for directed network by writing weight as a initial value of the matrix element e_{ij}

Each step of algorithm takes $\mathcal{O}(m + n)$ time and there are at most $n - 1$ nodes, so the worst-case time complexity is $\mathcal{O}((m + n)n)$ or $\mathcal{O}(n^2)$ on a sparse graph.
It testes with many random graphs of 128 vertices. The average number of edges within community is z_{in} and that of between communities is z_{out}. The total expected degree is $z_{in} + z_{out=16}$. It correctly classifies more than 90% of the nodes for till z_{out}

is less than 6. After that it begins to fail. When $z_{out} = 5$ the algorithm partitions 97.4% nodes correctly. For higher values of z_{out}, it performs much better than [11].

This algorithm was also tested against Zachary's karate club [27]. The maximum modularity obtained for this network is $Q = 0.381$ which divides the network into two communities of 17 nodes each. Only node 10 was not categorized correctly.

In case of American college football network, $Q = 0.546$ which is slightly less than best optimal value of $Q = 0.601$. The intra-conference games are more frequent than that of inter-conference. This gives an idea about the clusters the algorithm should find. The algorithm finds 6 communities instead of 11. Most of them corresponds to two or more conferences.

Extension of this is a fast-greedy algorithm proposed in [4]. It performs a greedy optimization and gives similar results compared to [20], but this is faster algorithm as it uses more sophisticated data structure. Its faster for real-world networks than original Girvan-Newman algorithm [11].

It stores adjacency matrix as an array and repeatedly merge matrices as two communities are merged. It does unnecessary computations while merging two community matrix elements when the network is sparse and value is 0 [20]. This happens many times as most of the elements of adjacency matrix of sparse graph are 0. To overcome this costly approach (space wise and time wise), authors of [4] have proposed new algorithm which uses some shortcuts and more sophisticated data structure.

To start with, they have defined two new quantities. One is

$$e_{ij} = \frac{1}{2m} \sum_{vw} A_{vw} \cdot \delta(c_v, i) \cdot \delta(c_w, j) \tag{7}$$

which is the fraction of edges linking vertices in community i to j. And δ-function $\delta(i, j)$ is 1 if i = j and 0 otherwise. Another is

$$a_i = \frac{1}{2m} \sum_{v} k_v \cdot \delta(c_v, i) \tag{8}$$

which is the fraction of end points of edges that are attached to vertices in community i. Using these values in equation of modularity, they have derived modularity as,

$$\begin{aligned} Q &= \frac{1}{2m}\sum_{vw}\left[A_{vw} - \frac{k_v k_w}{2m}\right] \cdot \sum_i \delta(c_v, i)\delta(c_w, i) \\ &= \sum_i \left[\frac{1}{2m}\sum_{vw} A_{vw} \cdot \delta(c_v, i) \cdot \delta(c_w, i) \right. \\ &\left. - \frac{1}{2m}\sum_v k_v \cdot \delta(c_v, i) \cdot \frac{1}{2m}\sum_w k_w \cdot \delta(c_w, i)\right] \\ &= \sum_i (e_{ij} - a_i^2) \end{aligned} \tag{9}$$

If the joining of two communities gives the larges change in Q then it merges those communities. Finding ΔQ is a slow process. To avoid that it keeps stores values of ΔQ into a matrix and updates it periodically. ΔQ_{ij} is calculated for only those communities which are joined by at least one edge. Each row of matrix is stored as a balanced binary tree and max-heap to get largest ΔQ in constant time.

It starts with assigning community to each vertex and calculating initial matrix of ΔQ and populating max-heap with largest element of ΔQ. Then it selects largest ΔQ from max-heap and combines community that corresponds to that value. After combining it updates the matrix and max-heap and increment Q by ΔQ.

The running time of this is $\mathcal{O}(md \log n)$ where d is depth of the dendrogram and $m,\ n$ are the number of edges and nodes. Many real world networks are sparse. For such networks, the complexity is $\mathcal{O}(n \log^2 n)$.

This approach was tested against the co-purchasing or recommender network from Amazon.com [4]. This can be represented as a directed graph in which nodes represent items and there is an edge between node A and B if B are frequently purchased by the buyers of A.The network has 409,687 nodes and 2,464,630 edges. The maximum value of Modularity is $Q = 0.745$ at which there are 1684 communities with average

size of 243 nodes each.

Most of the existing algorithms related to community detection require prior knowledge about communities. [23] proposed a new algorithm which finds the community structure of networks whose prior knowledge is unknown. It is a localized approach based on the label propagation.

The node belongs to which community is decided based on a community of majority of its neighbors. A unique label is assigned to each node. Iteratively, a node either accepts a new label that majority of its neighbors have or remains as it is. The order at which all nodes are updated is random. The updates are synchronous or asynchronous. In synchronous method, at t^{th} iteration a node i changes its label to labels of its neighbors at $(t-1)^{\text{th}}$ iteration.

$$C_x(t) = f(C_{x1}(t-1), C_{x2}(t-1), ..., C_{xk}(t-1)) \tag{10}$$

where C_x = label of node x at time t. And in asynchronous method, label of node is updated based on some of its neighbors who have already updated their label and those that have not yet updated in the current iteration.

$$C_x(t) = C_{x_{i1}}(t), C_{x_{i2}}(t), ..., C_{x_{i(m+1)}}(t-1), C_{x_{ik}}(t-1) \tag{11}$$

where x_{i1}, ..., x_{im} are the neighbors of x that have already been updated while $x_{i(m+1)}$, ..., x_{ik} are the neighbors that are not yet updated in the current iteration.

This process is continued till no nodes change their label. As this process continue, densely connected nodes adopt to same label and the no of labels decreases. At the end, the no of labels remain is equal to no of communities and nodes having same label are grouped together. Even though this process should be continued till no nodes change their label, in reality this does not happen. Because the algorithm breaks ties randomly, the labels of nodes keep changing even though their neighbors

remain same. To avoid that stops this process when every node has label which maximum its neighbors have. The algorithm follows these steps:

1. Assign unique label to each node. For given node x, $C_x(0) = x$
2. Set t = 1
3. Arrange nodes in random order and set it to X
4. For each node belonging to X, adopt a new label based on its neighbor. Break ties randomly.
5. If every node has a label which is same as the maximum no of its neighbors have, then stop the algorithm otherwise t = t + 1 and go to step 3.

They have tested this algorithm on various datasets like Zachary's karate club, Protein-protein interaction network, US college football network, World Wide Web(WWW) and Co-authorship network, Actor collaboration network.

Several different types of methods have been proposed in recent years. Most popular among these methods are divisive methods, agglomerative methods and optimization methods which maximizes or minimizes an objective function.

Fastest optimization algorithm was proposed in [4] which is a greedy approach of finding communities. It iteratively combines the two communities in order to maximize the network modularity Q (Equation 4). There are two drawbacks associated with this method. One is its tendency to generate large communities which includes almost all of the nodes. Further it cannot efficiently explore the hierarchical structure of the network. Another disadvantage is that it is super slow when applied to very large network.

Today almost all social networks have millions of users. So very large networks have

become computationally difficult to be executed using this algorithm [4]. Almost all large social networks have multilevel structure and thus there is a need to find methods that explore such hierarchical structure.

In this paper, Blondel et al. proposed an algorithm which finds communities with high modularity and explores the complete hierarchical structure, thus accessing different resolutions of community detection.

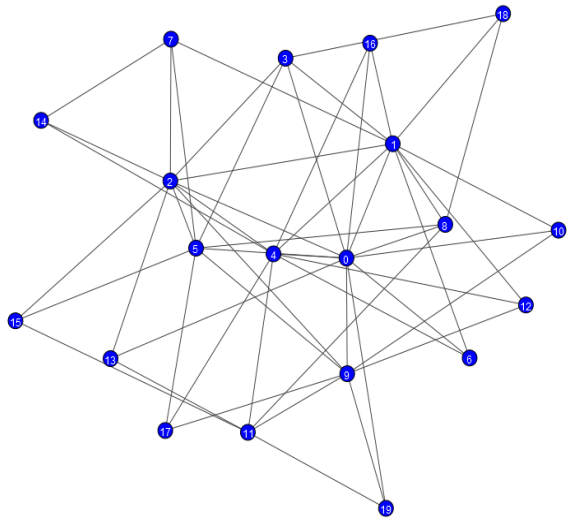
The method have 2 phases. In phase 1, every node is a separate community. Then the gain modularity is computed for every node i by considering each of its neighbors. The i is merged with the community of one of its neighbor if it maximizes the gain in modularity. ΔQ is calculated by merging an isolated node i into community C using formula,

$$\Delta Q = \Big[\frac{\sum_{in}+k_{i,in}}{2m} - \Big(\frac{\sum_{tot}+k_i}{2m}\Big)^2\Big] - \Big[\frac{\sum_{in}}{2m} - \Big(\frac{\sum_{tot}}{2m}\Big)^2 - \Big(\frac{k_i}{2m}\Big)^2\Big] \qquad (12)$$

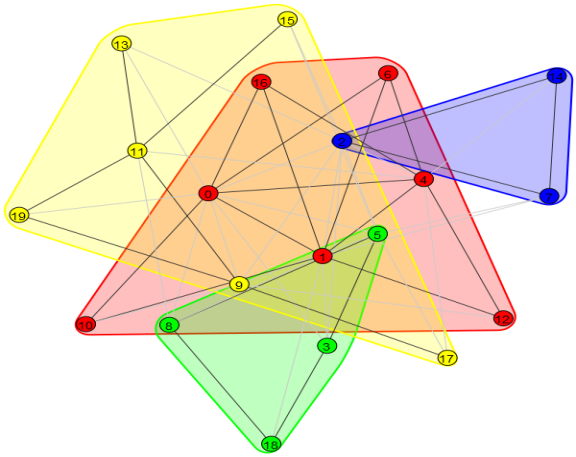
where $\sum_{in}$ = the sum of the weights of the links inside C, $\sum_{tot}$ = the sum of the weights of the links incident to node i, $k_{i,in}$ = the sum of the weights of the links from node i to the node in C and m = the sum of the weights of all links in the network. If ΔQ is negative, then i remains as it is. This process stops when no two communities can be merged further.

The phase 2 builds a meta network from the partitions obtained from phase 1. Every vertex gets a weight which is equal to the sum of weights of edges in corresponding community and the edges within communities are shown as self loops. Complete execution of phase 1 and 2 is a one pass. Input to the second pass is this meta network. This process stops when no changes can be done and the maximum Q is attained. At the end, the hierarchy of communities is acquired. Its example is shown in Fig.9. There are few advantages of this algorithm mentioned in the paper such as,

1. Easy to implement



(a) Input graph with 20 vertices and 54 edges



(b) 4 communities found by Louvain algorithm

Figure 9: Example of Louvain algorithm

2. Steps are intuitive

3. Outcome is supervised

4. Computationally fast

5. Solves resolution limit problem of modularity

Almost all the networks produced community structure with high modularity

and the computation time as well as the number of passes was small. In case of Karate club network, this algorithm was successfully completed in 3 passes. In the first pass, the number of communities was 6 and that was reduced to 4 in second phase and nothing happened in third phase.

They have also tested this algorithm on a synthetic network of 128 nodes. The network is partitioned into 4 communities of 32 nodes each. The probability for edges belonging to same community is P_{in} and that of between communities is P_{out}. The algorithm was evaluated based on the accuracy of correctly classifying nodes into respective communities. The algorithm produced results with accuracy 0.67 when $z_{out} = 8$, 0.92 when $z_{out} = 7$ and 0.98 when $z_{out} = 6$.

In case of the network of Belgian mobile phone company, this algorithm detected 6 levels of hierarchical communities. At the bottom, every customer account for their own community and at the top level 261 communities are found which account for 75% of all customers.

The modified version of Louvain algorithm is proposed by De et al.[7]. It presents an efficient community detection algorithm which detect communities in weighted large social networks. Also they are using K-path edge centrality algorithm to calculate EB.

This algorithm is divided into 3 steps:

1. To calculate k-path edge centrality
2. To calculate pairwise distance between nodes of the network
3. Use Louvain method to partition the network

The evaluation of partition in communities is attained by network modularity Q.

To calculate edge centrality, this paper is using Weighted Edge Random Walk k-Path

Centrality algorithm [6]. It calculates EB using the random walks of length k on an edge which propagates a message. It consists of 2 steps.

1. Nodes and edges weights are assigned
2. Simulation of message propagation

Initially weight is assigned to each node using its Local Effective Density. The local effective density $\delta(v)$ is calculated as,

$$\delta(v) = \frac{|I(v)| + |O(v)|}{2 \cdot |E|} \tag{13}$$

where $I(v)$ and $O(v)$ are in-going and out-going edges of node v. It represents how much a node contributes to the overall community structure. Similarly, weights on edges are calculated as,

$$w(e)^0 = \frac{1}{|E|} \tag{14}$$

$w(e)^0$ is called initial edge weight for edge e. Initially $|E|$ weight is equally distributed among all the nodes and this weight represents its initial rank.

In the 2^{nd} step of [6], ρ random walks of length k are exploited. This step further performs following operations,

1. A node v is selected based on its local effective density $\delta(v)$ as

$$P(v) = \frac{\delta(v)}{\phi} \tag{15}$$

   where $\phi = \sum_{v \in V} \delta(v)$ is a normalization factor.

2. All edges are marked as not traversed.
3. A method *messagepropagation* is called.

This method considers paths only of length k and avoids that message is not trapped into a loop. It selects an edge e_n with probability proportional to the edge weight $w(e_n)$,

$$P(e_n) = \frac{w(e_n)}{\gamma} \tag{16}$$

where $\gamma = \sum_{e_n \in \hat{O}(v_n)} w(e_n)$ is a normalization factor. It selects an edge e_n such that it is not previously traversed and it reaches node v_{n+1}. Then it gives $\beta = \frac{1}{|E|}$ to e_n, sets $T(e_n) = 1$ and increases the k-length counter. At the end, each edge is assigned with a centrality value (equal to its weight).

The edges are sorted in decreasing order of their centrality value. Then proximity between each pair of connected nodes is calculated using,

$$r_{ij} = \sqrt[2]{\sum_{k=1}^{n} \frac{(L^k(e_{ik} - L^k(e_{jk}))^2}{d(k)}} \tag{17}$$

where $L^k(e_{ik})$ is k-path edge centrality of edge e_{ik} and d(k) is the degree of the node. Higher the $L^k e_{ik}$, the more nearer nodes are [7].

The last step is to partition network based on proximities of nodes. This is done by Louvain method [2]. The partition of network ends when Q cannot be further improved. The time complexity of this algorithm is $\mathcal{O}(k|E| + \bar{d}(e)|V| + \gamma|V|) = \mathcal{O}(\Gamma|E|)$.

All these algorithms uses some centrality measures to find communities. There is one approach called a single linkage clustering which is one of the oldest method of hierarchical clustering. At each step, it combines two smaller clusters into one bigger cluster as long as vertices within two clusters are not combined yet. Most of the algorithms implementing this approach uses some distance function to decide which clusters are closer to each other and which are not.

In [24], they have mentioned that similarity of two different clusters is totally dependent on the similarity of two of it's closest vertices. Only those clusters which are

lot closer to each other than any other clusters are taken into account, and not those clusters which are at different part of the network.

The disadvantage of this approach is that it divides the vertices into clusters such that vertices belonging to different clusters are far from each other than that of vertices within same clusters.

One such algorithm is Single-link k-clustering algorithm . This is a greedy algorithm which divides vertices into k-clusters. The value of k is usually approximated depending on the dataset. This algorithm finds the closest pair of vertices and put each of the vertex into different cluster and adds an edge between them. This process is repeated $n - k$ times which result in exactly k clusters.

This closely corresponds to the MST of a weighted graph. If the MST is built on input graph then by removing just $(k - 1)$ edges the clusters can be seen. If used Kruskal's algorithm [14], then the stop condition would be when there are k connected components.

The betweenness centrality is an important factor in analyzing social networks, especially detecting communities. It ranks the actors by their importance in a network. Existing methods to compute betweenness centrality are computationally expensive for large networks as these methods find shortest paths between nodes. To overcome this drawback, [3] proposed an approach for large networks. It uses a new accumulation technique that works well with a single-source shortest-path algorithm and thus reducing time complexity[5].

[3] computes centrality by accumulating dependencies of each vertex instead of summing up pair-dependencies. It uses BFS for unweighted graphs and Dijkstra′s algorithm for weighted graph. It computes betweenness by solving single-pair shortest path algorithm. It computes dependency of vertex s on every other vertex v using

theorem proved in paper.

$$\delta_{s\bullet}(v) = \sum_{w:v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{s\bullet}(w)) \tag{18}$$

It starts with s and traverses the vertices in a decreasing order of a distance and keep accumulating dependencies. At the end, dependency of s on every other vertex is added to the centrality score of s. Same thing is done for every vertex. Its running time is $\mathcal{O}(mn)$ for unweighted graphs and $\mathcal{O}(mn + n^2 \log n)$ for weighted networks.

CHAPTER 4

New methods for community detection

4.1 Agglomerative Community Detection using Edge Betweenness

Here we describe the Agglomerative Community Detection using Edge Betweenness (ACUEB). For these experiments, we have used unweighted and undirected graphs. Also the implementation of this project is done using Python programming language and igraph library [18] which is a Python API for graphs and networks.
As explained in [2], Louvain algorithm is a two step greedy optimization algorithm in which first step is to iteratively merge smaller communities into bigger communities in order to maximize the gain in modularity Q (equation (4)). Second step forms a meta network from the communities of step I. Step I is applied to this meta network and this process is continued till there is a gain in the modularity.
In this experiment, we are using the meta network part of Louvain algorithm to find the hierarchical community structure. Our assumption is if the most important edges in the graph connects different communities, then the vertices within same communities must be closer and are connected by really small paths with less important edges. This experiment starts with assuming every vertex is an individual community. Iteratively, we find the edges with smaller EB ratio and add to the output graph, which is an input graph without the edges. The idea is to iteratively merge smaller communities into bigger communities and which community should be merged is not decided on the basis of gain in modularity but the shortest paths within vertices of that community. This process is continued till we achieve local maxima the modularity Q is achieved. At the end of this process, we find the underlying community structure with the highest modularity. We have further implemented two different methods for

this approach which are:

1. With meta-network
2. Without meta-network

4.1.1 With meta network

Let's $G = (V, E)$ be an input graph and $G' = (V)$ be an output graph. First, we find all edges with minimum EB value as shown in algorithm 3. First, this procedure calculates the EB of all the edges of G and then finds smallest EB. With this value it search for edges which are not already traversed or considered. If there are no such edges then edges for second-smallest EB values are searched.

Algorithm 3 Pseudo code to get edge(s) with minimum edge betweenness

Input: $G = (V, E)$, List of edges already traversed L, List of edge betweenness values already considered $EBList$
Output: List of edges with minimum edge betweenness ratio $EdgeList$
Procedure GET EDGES WITH MINIMUM EDGE BETWEENNESS(G, L, EBList)
$EdgeList \leftarrow \phi$
$E' \leftarrow$ Calculate edge betweenness for every edge e in E
$MinEdgeBetweenness \leftarrow min(E')$
$EdgeList \leftarrow$ Edges with the edge betweenness ratio $MinEdgeBetweenness$ which are not already traversed
end Procedure
return $EdgeList$

These edges are then added to G'. While adding this list of edges $EdgeList$ to G', the end points of each edge are looked up in original input graph G. It is possible that end points of an edge are inside community and hence represented by community index instead of their original index. It is also possible that only one of the end point

is inside existing community. In such cases, it is important to find the actual end points $source, target$. These $source, target$ are found using procedure 4.

Algorithm 4 Pseudo code to get actual end points of an edge

Input: $G(V, E)$, Membership vector M, Edge e, List of edges already traversed S
Output: End points of edge e

```
Procedure GET ACTUAL ENDPOINT OF EDGE (G, M, e, S)
    source ← End point 1 of e
    target ← End point 2 of e
    for each vertex v1 in M do
        if v2 = source then
            for v2 in M do
                if v2 = target and (v1, v2) are connected in original input graph and
edge connecting (v1, v2) is not already traversed and e = edge(v1, v2) then
                    source ← v1
                    target ← v2
                    break
                end if
            end for
        end if
    end for
end Procedure
return source, target
```

Algorithm 5 and 6 find all the connected components, let's say S, in G'. For every vertex $v \in V$, it calculates it's adjacent neighbors using BFS algorithm, explained in Figure 2.1. BFS traverses through the breadth of the graph such that it explores the direct neighbors of starting vertex v, then the direct neighbors of neighbors of that vertex and so on. This way it finds a connected component c if given a starting vertex v. An individual vertex with degree 0 is a connected component with only one element.

Algorithm 5 Pseudo code for finding all connected components

Input: $G = (V, E)$, no of vertices in input graph n
Output: List of all connected components S

```
Procedure GET ALL CONNECTED COMPONENTS(G, n)
    S ← {}
    T ← {} Vertices traversed so far
    while every vertex v is not visited do
        S' ← GET CONNECTED COMPONENT USING BFS (G, v)
        for each vertex v' in S' do
            T ← T ∪ v'
            S ← S ∪ S'
        end for
    end while
end Procedure
return S
```

Once connected components S are looked up then a vertex cluster object *vertexCluster* is created. A vertex cluster is object in igraph [18] which specifies to which partition each vertex belongs to and also the modularity of the partition. This object is created using connected components found in G'.

Algorithm 6 Pseudo code for a connected component using BFS

Input: $G = (V, E)$, Starting vertex v
Output: A connected component c

```
Procedure GET CONNECTED COMPONENT USING BFS(G, v)
    c ← BFS (G, v) (Using algorithm 1)
end Procedure
return c
```

A membership vector M is initialized. This vector stores the index of the community in which that vertex is partitioned. For every component c in S, we look up for such an index, using igraph's $min()$ function, which is a minimum vertex index and used as the community index. That is, all the vertices within that

component are identified by that minimum index. Similar to $min()$, igraph provides many function such as $max(), sum(), mean(), median(), product()$ et al. which are used to select index which will represent all vertices in that component.

If there is only one vertex in c then we assign that component as an individual community. If the component has more than one vertices then for each vertex an entry is made in a membership vector M specifying the community index. From this M, a *vertexCluster* object is created. This vertex clustering object has list of all communities C for this iteration.

Algorithm 7 Pseudo code to get communities from connected components

Input: $G = (V, E)$, List of connected components S
Output: Vertex clustering object *vertextCluster*

```
Procedure GET VERTEX CLUSTERING FROM CONNECTED COMPONENTS(G, S)
    Vertex membership vector M ← φ
    for each component c in S do
        Vertex v' ← Minimum vertex index in component
        for each vertex v in component do
            Membership of vertex M(v) ← v'
        end for
    end for
    vertextCluster ← Communities using M
end Procedure
return vertextCluster
```

We calculate the modularity Q' for the community set C and compare with existing modularity Q. After that, We are building a meta network, using igraph's built-in method, in G out of the communities found in G' i.e. the group of vertices which belong to one community in G' is represented by only one vertex in G. This is repeated for every cluster of vertices.

Once this is done, then we again find edges with smallest EB values and then add to the output graph G' and so on. This is continued till we achieve local max-

ima for the modularity Q. The steps of this algorithm are summarized in Algorithm 8.

Algorithm 8 Pseudo code for ACUEB to find communities with a meta network

Input: $G = (V, E)$
Output: Graph G' having a set of communities C of maximum modularity Q
Procedure ACUEB WITH META NETWORK(G)
Number of vertices $n \leftarrow |E|$
Maximum modularity $Q \leftarrow 0$
$G'(V, E') \leftarrow G$ without edges E
List of edges already traversed $L \leftarrow \phi$
List of edge betweenness values already considered $EBList$
while local maxima not achieved **do**
List of edges with minimum edge betweenness $edgeList \leftarrow$ GET EDGES WITH MINIMUM EDGE BETWEENNESS (G, L, $EBList$)(Using algorithm 3)
for each edge e in edgeList **do**
End points of edge($source, target$) $\leftarrow$ GET ACTUAL ENDPOINT OF EDGE(G, M, e, L)
Add e to G'
end for
Connected component $C \leftarrow$ GET ALL CONNECTED COMPONENTS(G, n)
$vertxCluster \leftarrow$ GET VERTEX CLUSTERING FROM CONNECTED COMPONENTS(G, C)
$G \leftarrow$ Build a meta network from C
$Q' \leftarrow$ Modularity(C)
if Q' is better than Q **then**
$Q \leftarrow Q'$
end if
end while
end Procedure

4.1.2 Without meta network

This approach is mostly similar to above approach except we are not building a meta-network here. For this approach also, we start with each vertex being in an individual community. We then explore the list of edges $edgeList$ with smallest EB values and add those to G'. Once these edges are added, we are not finding

connected components or building meta networks as we did in 4.1.1, but we are directly calculating the modularity Q' of the formed communities. We are continuing this till we achieve local maxima.

Algorithm 9 Pseudo code for ACUEB to find communities without a meta network

Input: $G = (V, E)$
Output: Graph G' having a set of communities C of maximum modularity Q

```
Procedure ACUEB WITHOUT META NETWORK(G)
    Maximum modularity Q ← 0
    G'(V, E') ← G without edges E
    List of edges already traversed L ← φ
    List of edge betweenness values already considered EBList
    while local maxima not achieved do
        List of edges with minimum edge betweenness edgeList ← GET EDGES
WITH MINIMUM EDGE BETWEENNESS (G, L, EBList)(Using algorithm 3)
        for each edge e in edgeList do
            Add e to G'
        end for
        Q' ← Modularity(G')
        if Q' is better than Q then
            Q ← Q'
        end if
    end while
end Procedure
```

4.2 Single-link k-clustering algorithm using NOVER

To conduct this experiment, we are using weighted undirected graphs. In this experiment, we are trying to find communities based on the concept of single link clustering (chapter 3). The gap between vertices of different clusters uses NOVER measure (Equation (3)) i.e. weights on the edges are calculated using NOVER.
The pseudo code for this experiment is summarized in algorithm 10. Let's assume $G = (V, E)$ is an input graph. First step is to calculate NOVER measure for every edge $e \in E$. Then we are building a MST using Kruskal's algorithm which uses a

priority queue to sort the edges by their weights and removes self loops and cycles. Once the MST $G' = (V, E')$ is built, we need to approximate the values of k.

Algorithm 10 Pseudo code for community detection using Single-link k-clustering algorithm using NOVER

Input: $G = (V, E)$
Output: Set of communities C with maximum modularity Q

```
Procedure COMMUNITIES USING KRUSKAL'S ALGORITHM(G)
    C ← φ
    for each edge in E do
        Calculate Neighborhood Overlap (NOVER) np score using Equation 3
        edge.weight ← np
    end for
    G' (V, E') ← KRUSKAL (G) (From algorithm 2)
    for each e in E' do
        Calculate edge betweenness of e using Equation 2
    end for
    Initialize maximum modularity Q to 0
    k ← random value between 0 and |E'|
    while Q is not maximum do
        Remove (k − 1) edges from G' in descending order of edge betweenness ratio
        C' ← community structure after removing (k − 1) edges
        Q' ← modularity of C'
        if Q' > Q then
            Q ← Q'
            C ← C'
        end if
        Approximate value of k based on k, 2k, k/2
    end while
end Procedure
return C
```

We calculate the EB values for every edge e in E'. This helps in deciding which edges are important in the G' as well as the actual EB ratio decides how close the end points of that edge are. The edges are eliminated in descending order of EB values. We start with a random value of k such that $0 \leq k \leq |E'|$. After removing the $(k-1)$

edges based on their EB values, we get the community structure with k number of communities which results in modularity Q'. We also check community structure and the modularity for $2k$ and $\frac{k}{2}$. The highest value among those two will be assigned to k for the next iteration. This process is continued till local maxima for the modularity Q is achieved. At the end of this procedure, we get to see the hierarchical community structure.

In the next chapter, the experiments and results are specified which will help in understanding these methods more clearly.

CHAPTER 5

Experimental Results

In this chapter, we are evaluating our methods (see chapter 4) using experiments on several real-world and synthetic networks and compare them against the existing ones, namely:

1. Louvain algorithm [2]
2. Girvan-Newman algorithm [11]

These two algorithms are already implemented in igraph library. We have compared all the algorithms based on two factors. One is the modularity Q and second is the number of communities found.
Some of the data sets and models that we have used are:

1. Albert-Barabasi model
2. Zackary's karate club
3. Facebook network

The data models and results when used that model are explained in following sections.

5.1 Synthetic network models

5.1.1 Albert-Barabasi model

We have used one synthetic random graph model which is Albert-Barabasi model [1]. This model is a scalable, random and robust graph model which

provides synthetic networks. These networks have a power law distribution property. Power-law distribution property specifies only few vertices in a network have highest degree. We have considered a random model of 30 vertices and 84 edges which is shown in Fig10. Table 1 displays comparison between the 5 algorithms.

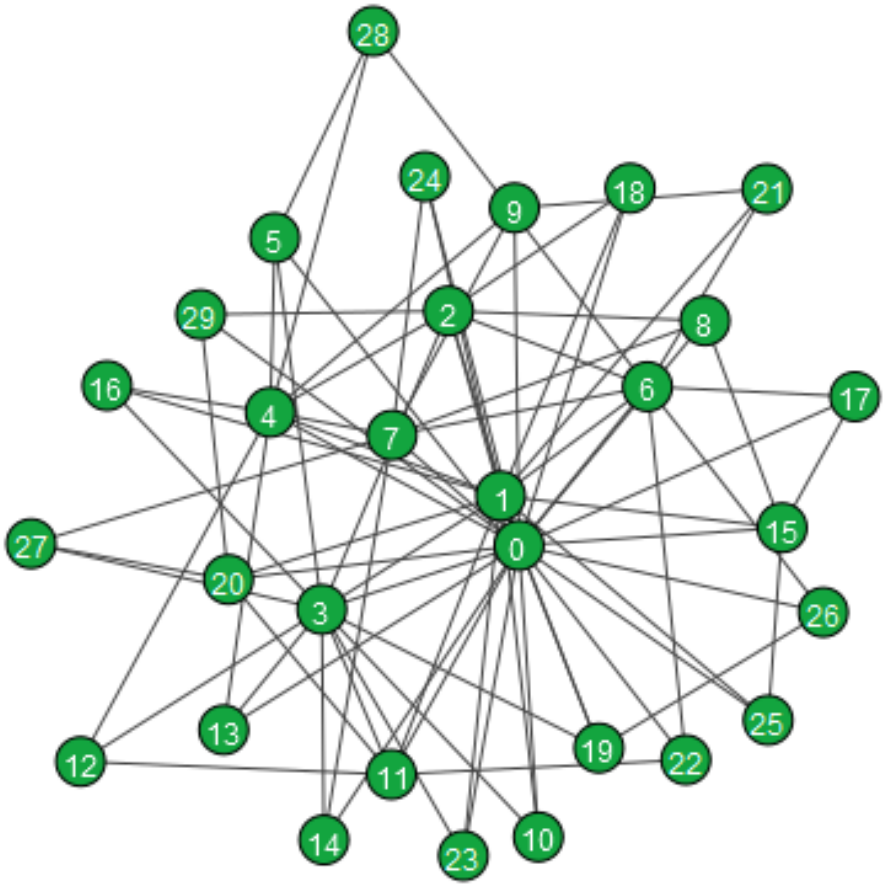


Figure 10: Barabasi graph with 30 vertices and 84 edges

Table 1: Comparison of the algorithms for the Barabasi model

Algorithm	No of communities	Modularity (Q)
Louvain algorithm	5	0.207
Girvan-Newman algorithm	9	0.074
ACUEB with meta network	18	0.25
ACUEB without meta network	16	0.6
Single-link k-clustering algorithm	7	0.73

The Girvan-Newman and Louvain algorithms for this graph model generates 9 communities with $Q = 0.074$ and 5 communities with $Q = 0.207$ respectively. Fig.11 and Fig. 12 show their communities. When the ACUEB with meta network uses this graph model, then we see that it finds 18 clusters with the modularity $Q = 0.25$. This is shown in Fig.13. We used this graph model for the ACUEB without meta network then we observed that community structure and the modularity was better than the rest of the algorithms. We found 16 clusters with the modularity $Q = 0.6$ as shown in Fig.14.

The MST built using Kruskal's algorithm for this dataset is shown in Fig.15. Table 2 shows different values of k and respective values of Q when we remove $k - 1$ edges. All 3 community structures are shown in Fig.16 and Fig.17. This algorithm is also giving better solution when compared against the rest of them.

Table 2: Communities with different values of k for the Barabasi graph model

K	No of communities	Modularity (Q)
4	4	0.57
7	7	0.0.73
14	14	0.60

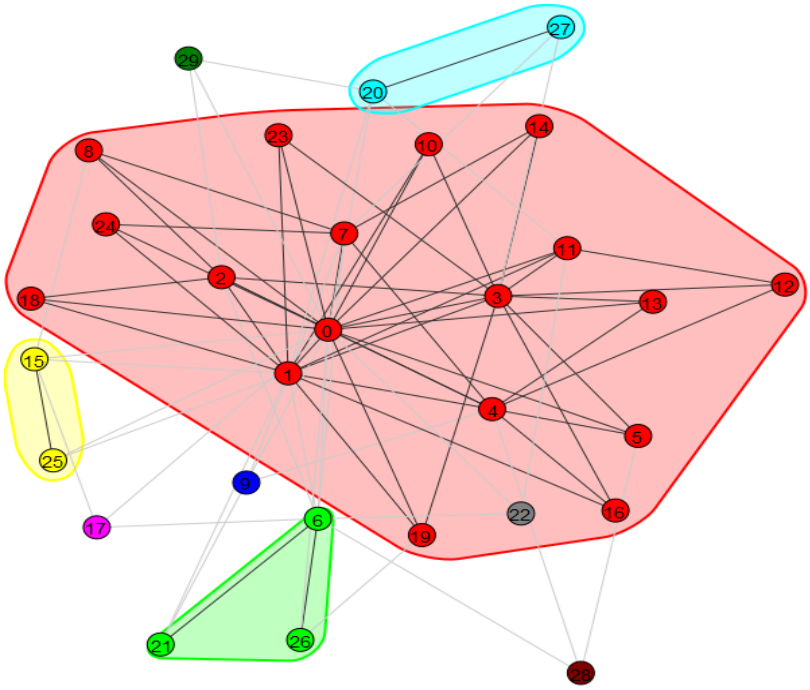


Figure 11: Output graph of the Girvan-Newman algorithm for the Barabasi graph model

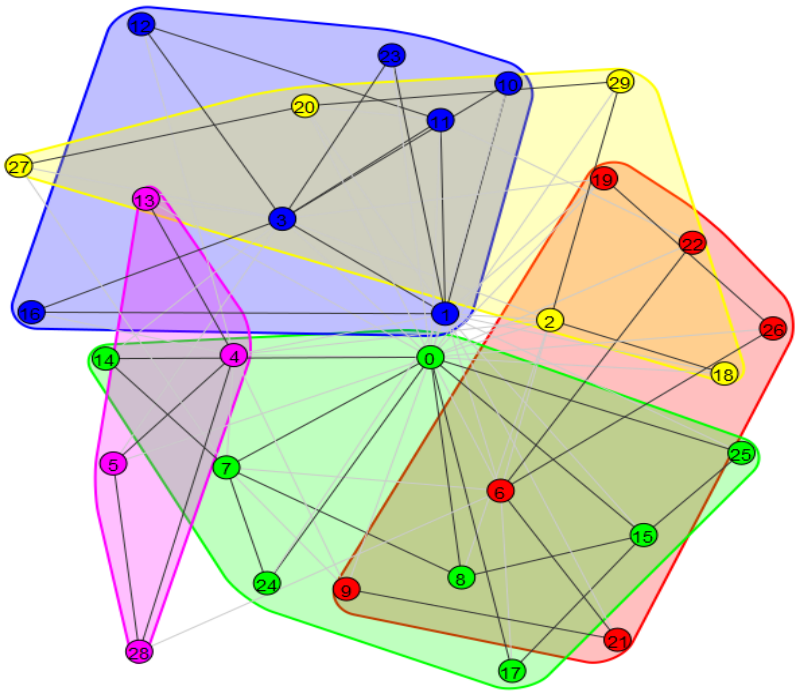


Figure 12: Output graph of the Louvain algorithm for the Barabasi model

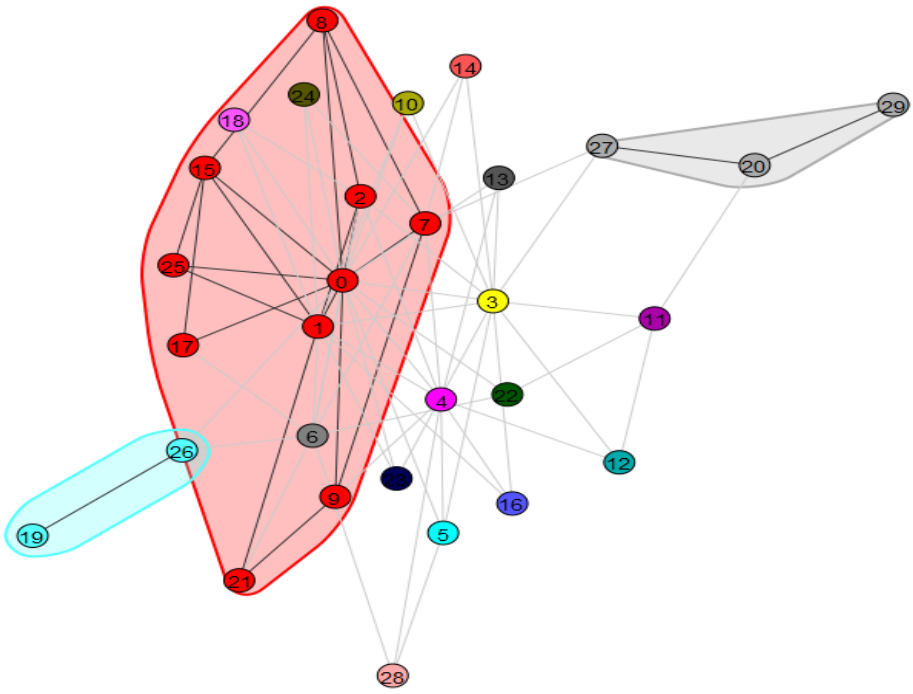


Figure 13: Clusters found by ACUEB with meta network

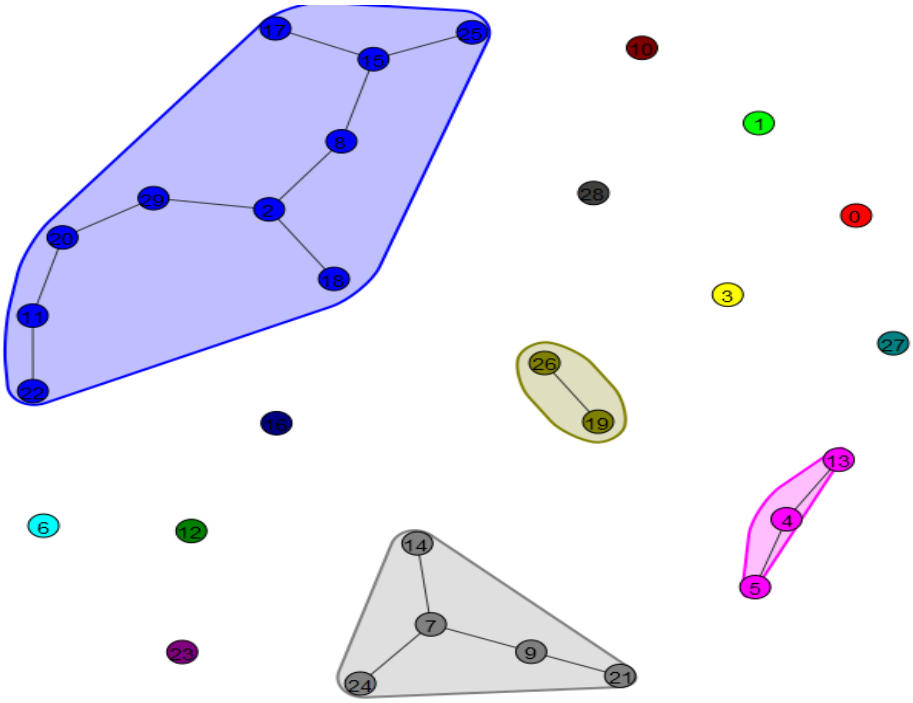


Figure 14: Communities detected by ACUEB without meta network

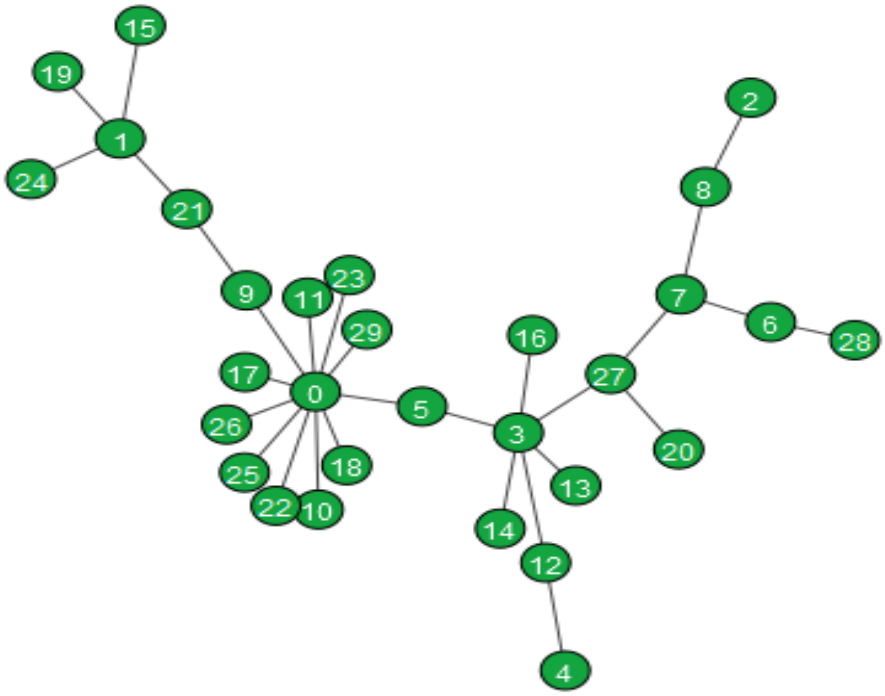


Figure 15: MST built for barabasi model

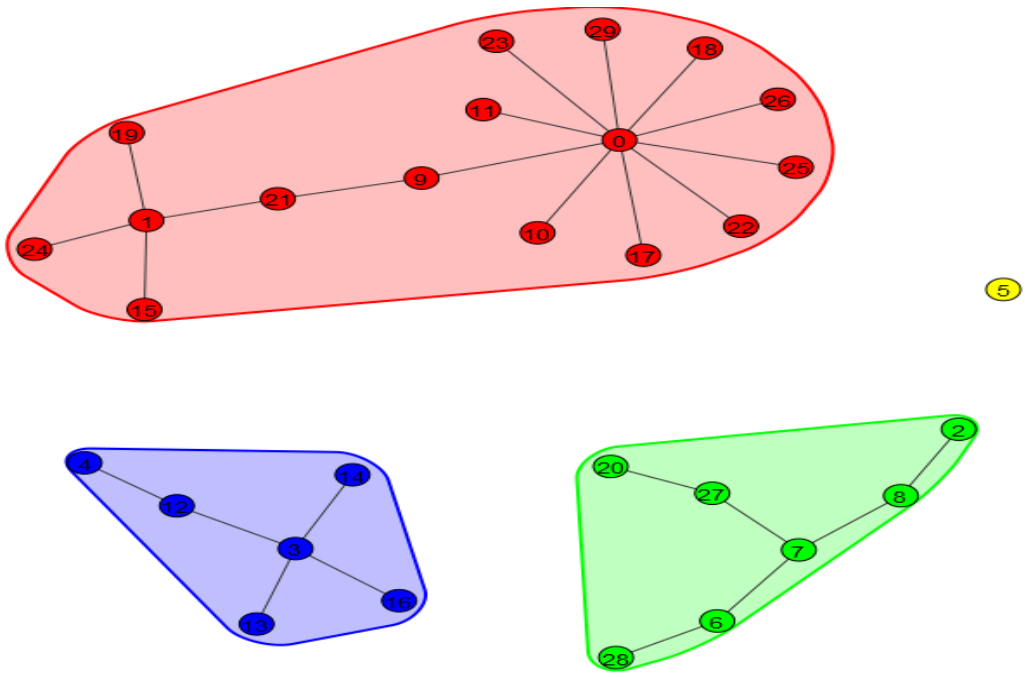
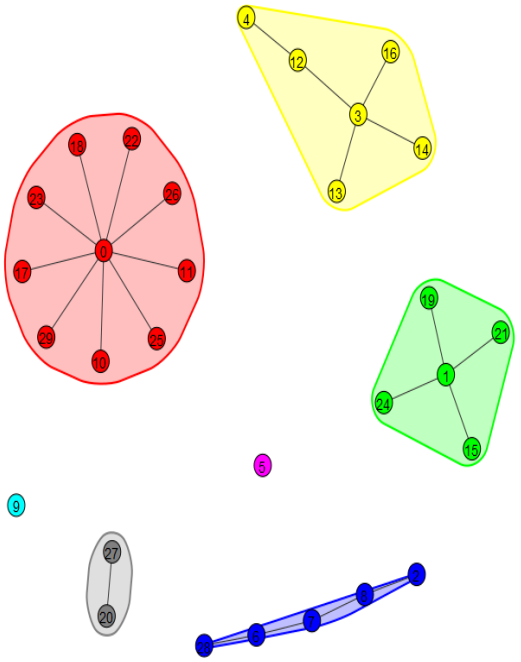
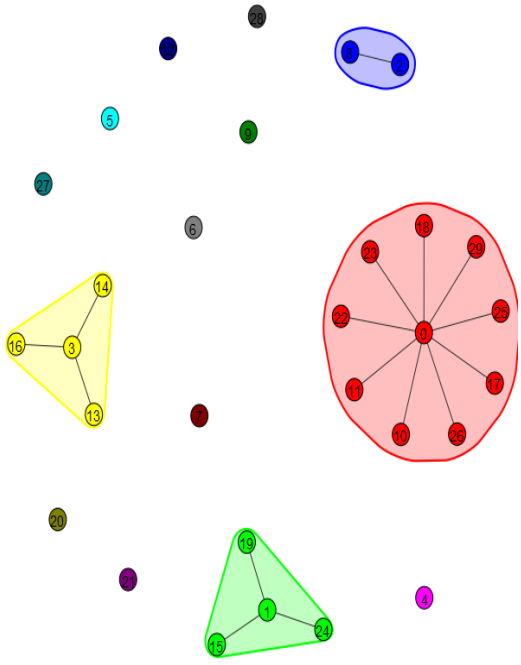


Figure 16: Communities detected by the single-link k-clustering algorithm with $k = 4$



(a) k=7



(b) k=14

Figure 17: Communities detected with different values of k for the Barabasi model

5.2 Real-world networks

Synthetic, computer generated random models give controlled test cases. So, it is really important to test these algorithms on real datasets.

5.2.1 Zachary's Karate Club

It is a famous network model of the observations Zachary explained in his study [27]. This dataset is plotted in Fig.18. Over the course of 2 year, he monitored members of a karate club and relationship between those members. Over time, dispute arose between team's instructor and administrator which resulted in club splitting in two separate clubs. Half of the original club joined the new club [11]. Zachary build a network model based on his observations which included 34 vertices and 78 links.

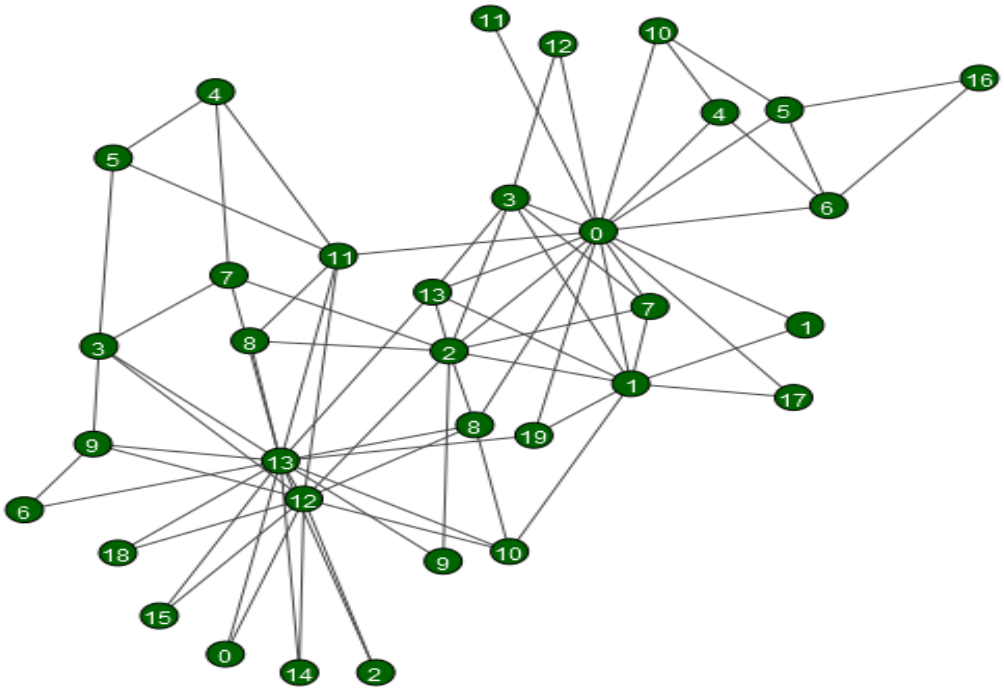


Figure 18: Zachary's karate club

Table 3 explains the results for each algorithm when we used this dataset. The expected result for this dataset is to have 2 communities each representing instructor (vertex 33) and administrator (vertex 0). Thus, communities around these vertices are formed. The Louvain algorithm [2] finds 4 communities for this dataset

Table 3: Comparison of the algorithms for the karate club dataset

Algorithm	No of communities	Modularity (Q)
Louvain algorithm	4	0.418
Girvan-Newman algorithm	4	0.401
ACUEB with meta network	13	0.498
ACUEB without meta network	12	0.588
Single-link k-clustering algorithm	2	0.5

with $Q = 0.418$. The communities are shown in Fig.19. As you can see from the image, almost all of the vertices are grouped around vertex 0 and 33. Even though this dataset has 2 communities, both Girvan-Newman algorithm finds 4 communities.

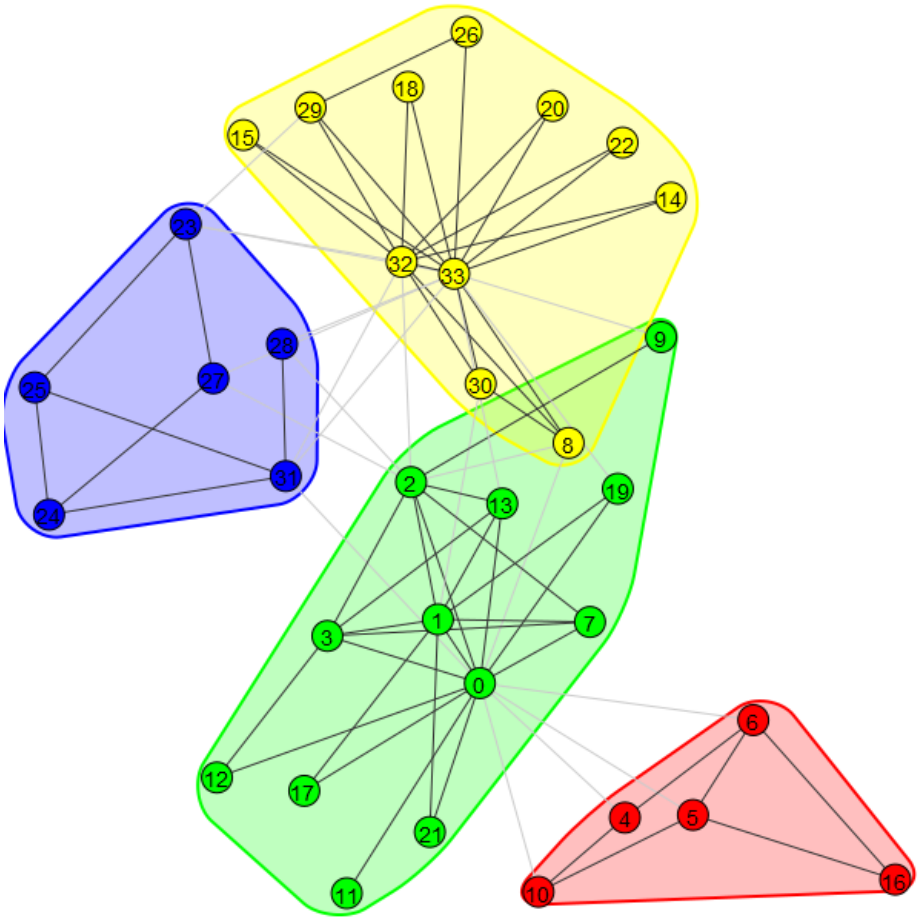


Figure 19: Communities structure found in the Karate club using the Louvain algorithm

Figure 20 shows the communities detected by ACUEB with meta network. There are total 13 communities with Q =0.498. We can see that there are two large chunk of vertices which are clustered around vertex 0 and 33. The communities detected using ACUEB algorithm without meta network are shown in Fig.21. It has detected 12 communities with Q =0.588 which is much better that the Louvain and the Girvan-Newman algorithms. Although both Fig.20 and Fig.21 shows 2-3 prominent chunks of vertices grouped together, there are some vertices which are not clustered in any communities and are considered as a separate communities.

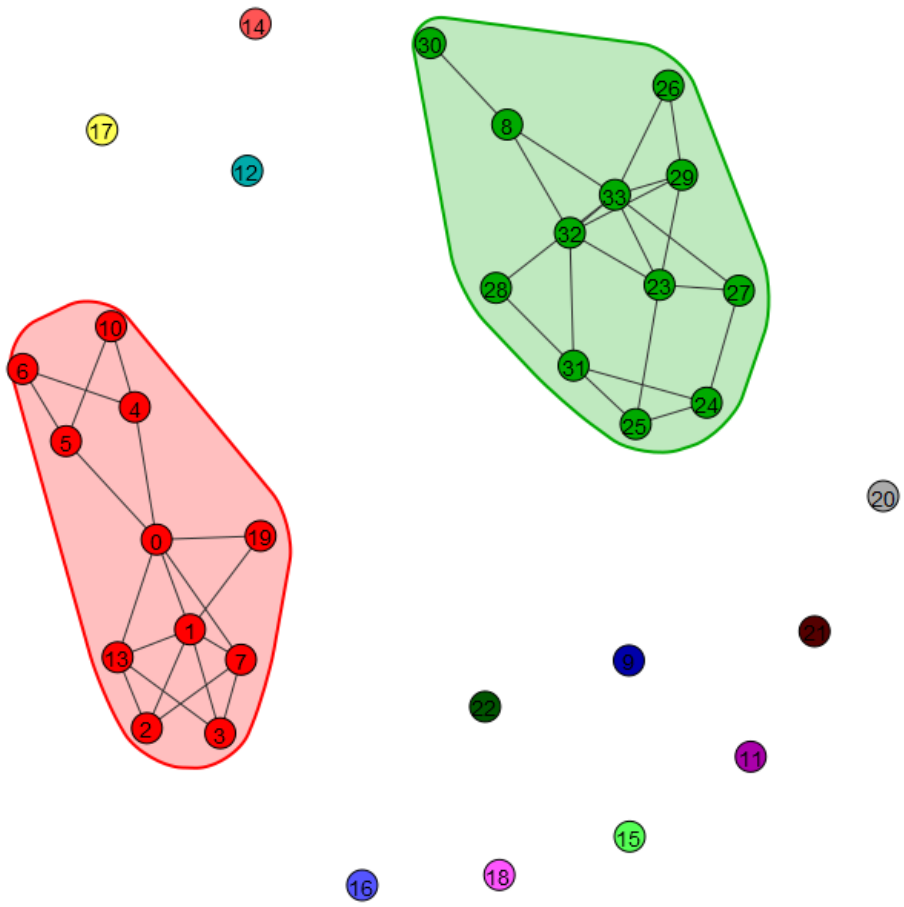


Figure 20: Communities detected using ACUEB algorithm with the meta network

The Single-link k-clustering algorithm using NOVER gives the best result among these 5 algorithms for this dataset. The MST obtained by running Kruskal's algorithm is shown in Fig.22.

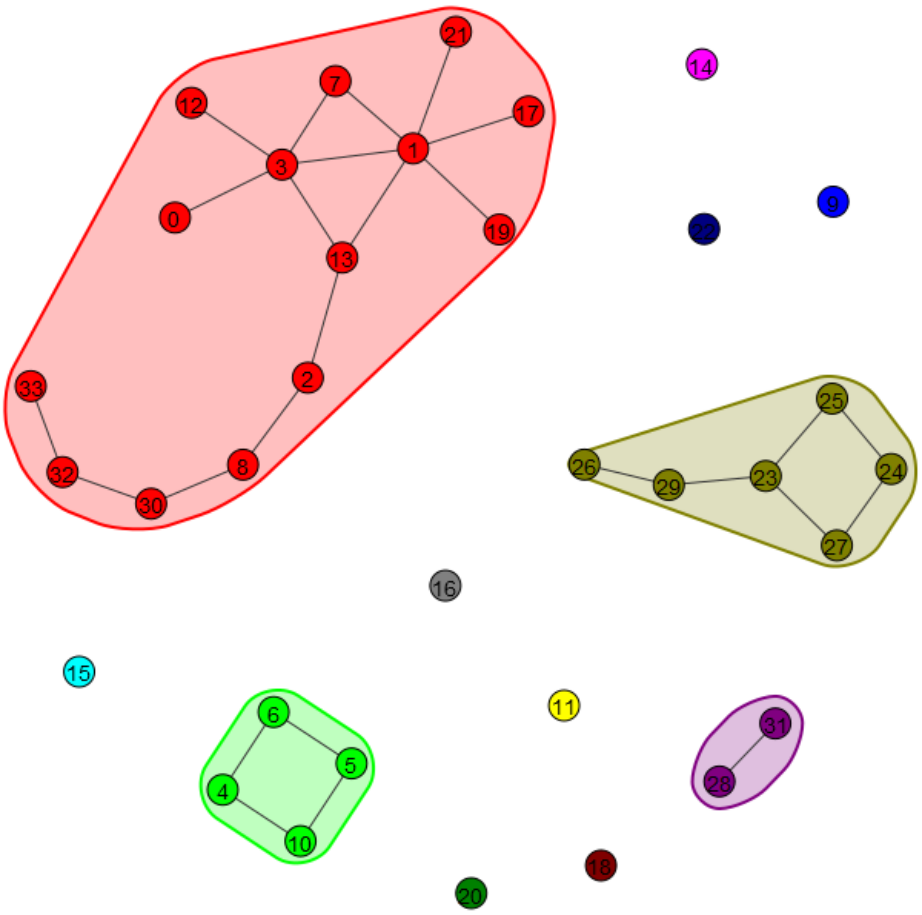


Figure 21: Communities detected using ACUEB algorithm without meta network

Table 4 shows different values of k and the community structure that we get by removing $k-1$ edges from MST. As we can see, as the value of k decreases, the modularity Q also decreases. Communities for all these values of k are shown in Fig.23.

Table 4: Communities with different values of k for the Karate club dataset

K	No of communities	Modularity (Q)
2	2	0.5
4	4	0.682
8	8	0.724
16	16	0.722

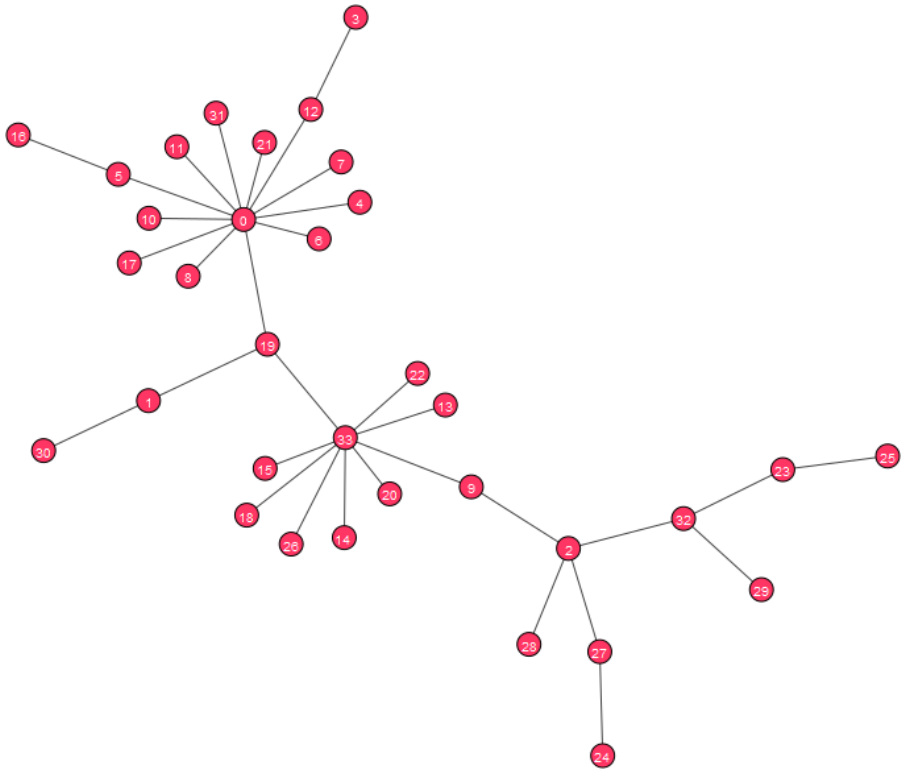
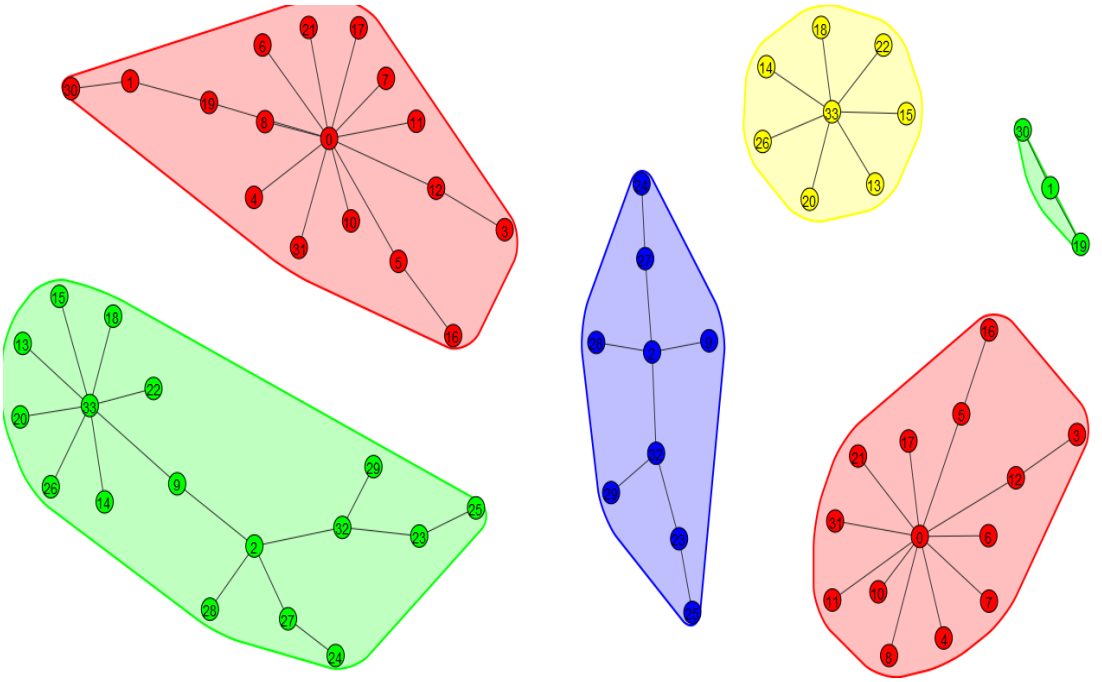
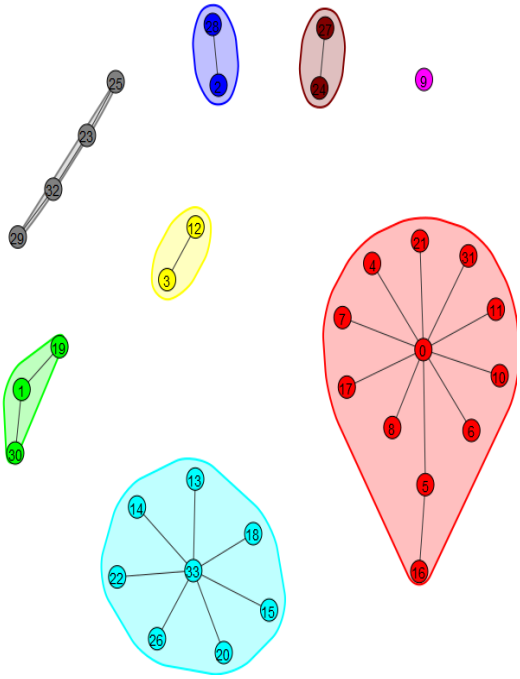


Figure 22: MST built for the karate club dataset

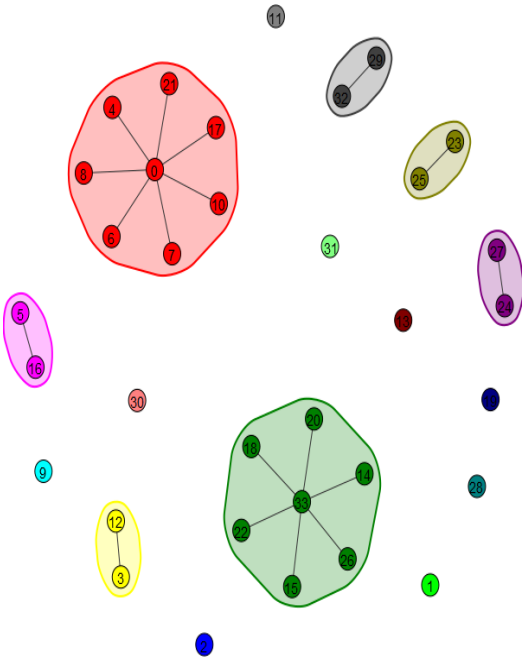


(a) k=2

(b) k=4



(c) k=8



(d) k=16

Figure 23: Communities detected with different values of k for the karate club dataset

5.2.2 Facebook friendship network

This is a friendship network between Facebook users [9, 16]. In this network the vertex represents a user and an edge indicates that users represented by the end-points are friends. This is an unweighted and directed network with 2888 vertices and 2981 edges. For the purpose of this project, we have ignored the directions of the edges. The friendship network is shown in Fig.24. It is clear from the figure that this network is not much dense i.e. some of the vertices are too far from each other as compared to other.

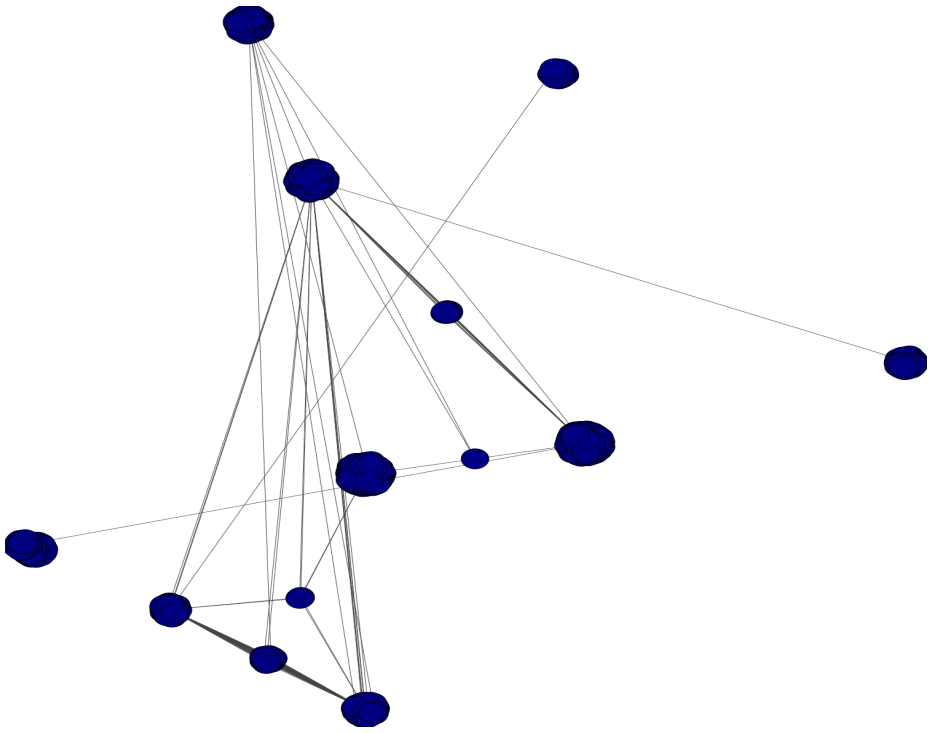


Figure 24: Facebook friendship network

Table 5 shows the comparison between algorithms based on the modularity Q. 3 of these algorithms are getting similar results with almost similar modularity.

Table 5: Comparison of the algorithms for the Facebook friendship network

Algorithm	No of communities	Modularity (Q)
Louvain algorithm	8	0.8
Girvan-Newman algorithm	8	0.80
ACUEB with meta network	18	0.735
ACUEB without meta network	17	0.639
Single-link k-clustering algorithm	18	0.82

The Louvain algorithm finds 8 communities in this friendship network. The modularity is $Q = 0.8$. Girvan-Newman algorithm also finds 8 clusters with same modularity as that of Louvain algorithm. Both of these approaches are displayed in Fig.25 and Fig.26.

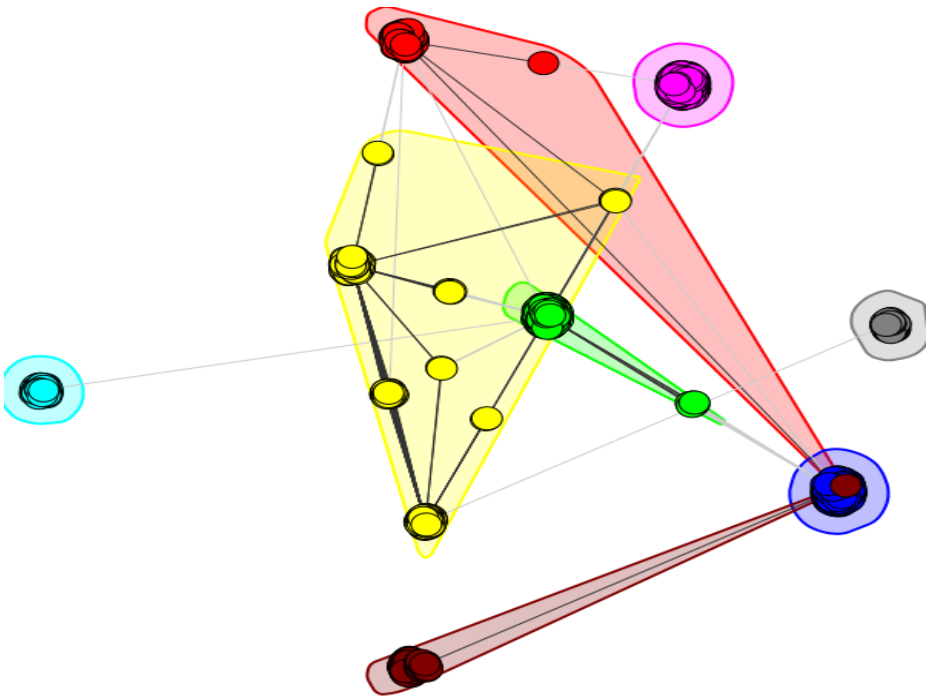


Figure 25: Output of the Louvain algorithm for the Facebook friendship network

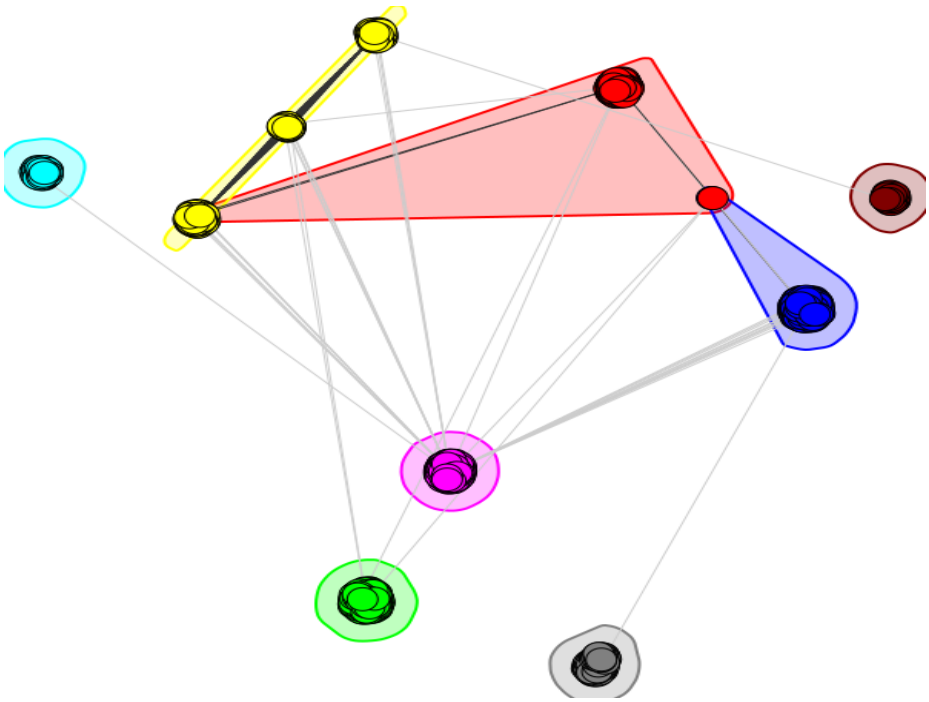


Figure 26: Output of the Girvan-Newman algorithm for the Facebook friendship network

Both ACUEB algorithm with meta network and single-link k-clustering algorithm finds 18 communities. But the modularity for the one with meta network is $Q = 0.735$ (Fig.27) and that of single-link k-clustering algorithm is $Q = 0.82$. The MST for this friendship network is shown in Fig.28 and it's communities are shown in Fig. 29. Fig.30 shows 17 clusters grouped together with the modularity $Q = 0.639$. From all these communities we can see that some clusters are at very less distance from each other than that of remaining ones.

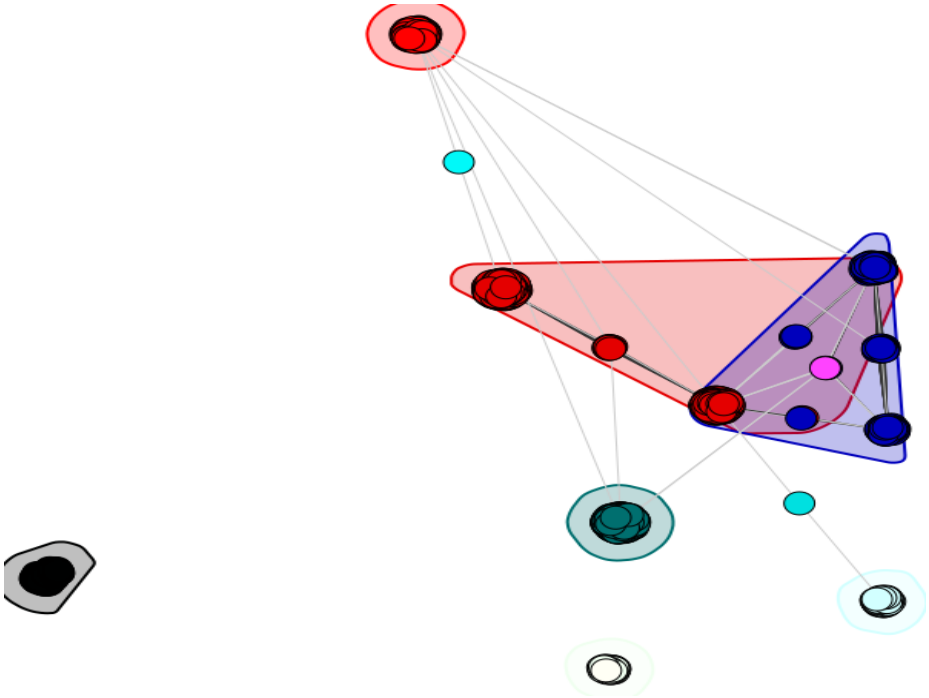


Figure 27: Communities detected by ACUEB algorithm with the meta network in Facebook friendship network

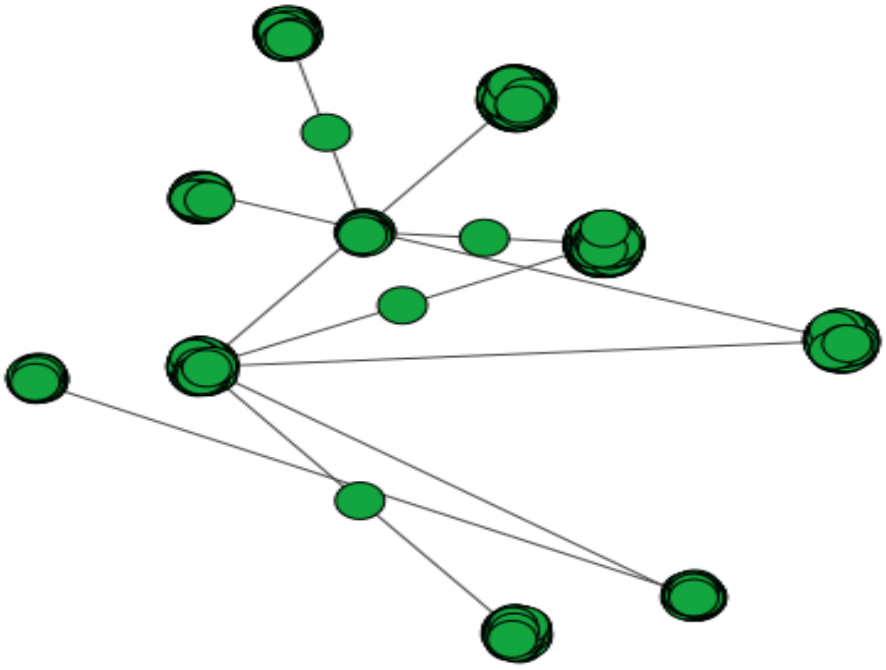


Figure 28: MST of Facebook friendship network

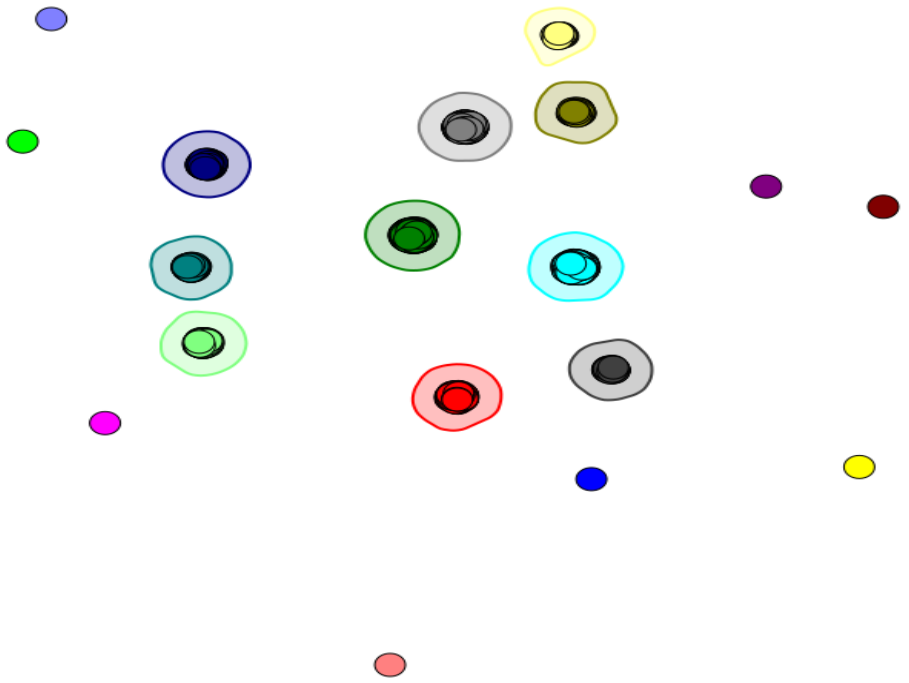


Figure 29: Communities detected by the single-link k-clustering algorithm in the Facebook friendship network

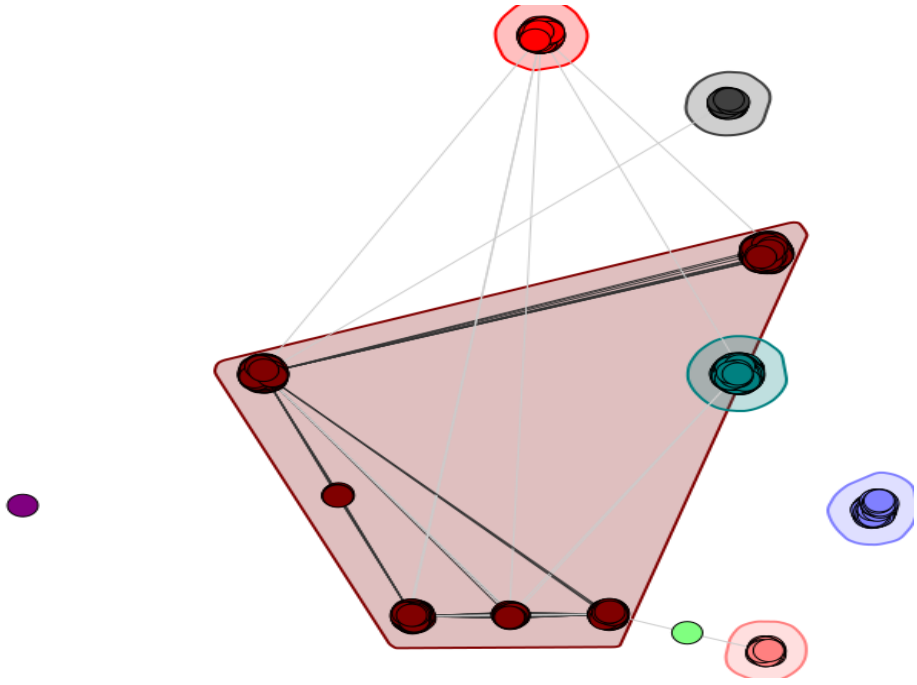


Figure 30: Communities detected by ACUEB algorithm without meta network in the Facebook friendship network

CHAPTER 6

Applications

For last few years the analysis of social networks has gained substantial attention. Studying the people, their behavior and relationships have application in many fields like biology, anthropology, information science, history, politics, psychology economics etc [26].

Community detection is one of the approach which will be helpful in analyzing these networks. This is because it is a human tendency to group together things which are co-related. Almost every social networks has some elements which can be clustered. In case of protein-protein synthesis network, it is very important to group together the cells which perform similar functions. Distributed architectures have multiple machines running simultaneously, each performing one type of task. Machines performing same task can be clustered together so as to analyze, maintain its architecture.

Another such application is the building of smart cities. A smart city is a well planned area where they use latest information technology tools to efficiently use resources, solve the problems and make city a better place to live. Building such cities is a very slow and step-by-step process as the problems and issues city faces need to be identified.

Citizen play an important role in identifying the problems. People must voice their opinion on various issues as well as share ideas to improve city planning management [12]. There are many social platforms like Facebook, Twitter and more which let people effectively share opinions, pictures, comments. We can group these people by their opinion, common issue or sentiments to form different communities. In each

community, some people are more active in voicing their opinion. They act as influencers in their network and the opinions of the rest of the people in that network is influenced by such influencers.

In [12], the authors have proposed a novel approach to find such communities and find the influential people in the network. Finding such influential individuals in a community will influence the community at large. This will benefit in campaigns of spreading awareness, increase public participation in the planning process etc.

These algorithms and methods assist in examining very large social networks. Recommendations, epidemic spreads, link predication, network communications, chemical chain reactions, business intelligence etc. are some other examples where the community detection is a vital factor.

CHAPTER 7

Conslusion

In this project, we propose three different approaches to find communities in a network. In our first approach of the agglomerative clustering algorithm with meta network, we are finding the communities by iteratively combining smaller ones into bigger ones using edge betweenness measure. At each iteration, we are also finding the meta network for formed communities. This meta network is helpful in finding communities within communities. The stop condition for this algorithm is to find the clusters with maximum modularity Q.

ACUEB algorithm without meta network also finds the community using the edge betweenness, but it does not make use of meta network. The edge betweenness is not re-calculated which improves the performance of this algorithm as compared to others.

In our last approach we propose a single-link k-clustering algorithm using Neighborhood overlap (NOVER). This approach builds a MST using Kruskal's algorithm and removes $k-1$ edges to find the underlying community structure.

At last, we carry out experiments using different synthetic and real-world networks. These experiments successfully compares and evaluates the performance of the proposed algorithms with known algorithms in terms of the number of communities and the modularity Q. All three proposed algorithms performs really well when compared with the Girvan-Newman and Louvain algorithm.

As a future work, we plan to compare these algorithms using different objective criteria to evaluate the performance. There are various known methods which compare the partitions in a network like the variation of information, computational cost etc.

Latest development in parallel computing, distributed architecture, big data, super computing is making it really easy to study, inspect and evaluate social networks efficiently. All these are well known methods to find communities. However, there are many other techniques to detect them like machine learning algorithms.

Finally, the community detection covers only small percentage of methods using which we can analyze these social trends, relationships, and behavior. There are many other means by which we can store, retrieve and analyze this large amount of information.

Bibliography

[1] Albert-László Barabási and Réka Albert. "Emergence of scaling in random networks". In: *science* 286.5439 (1999). Accessed on 2017-03-15, pp. 509–512. URL: https://arxiv.org/pdf/cond-mat/9910332.pdf%3Forigin%3Dpublication_detail.

[2] Vincent D Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of statistical mechanics: theory and experiment* 2008.10 (2008), P10008. URL: http://iopscience.iop.org/article/10.1088/1742-5468/2008/10/P10008/meta.

[3] Ulrik Brandes. "A faster algorithm for betweenness centrality*". In: *Journal of mathematical sociology* 25.2 (2001), pp. 163–177. URL: https://kops.uni-konstanz.de/bitstream/handle/123456789/5739/algorithm.pdf?sequence=1.

[4] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. "Finding community structure in very large networks". In: *Physical Review E* 70.6 (2004), p. 066111. URL: http://journals.aps.org/pre/abstract/10.1103/PhysRevE.70.066111.

[5] Thomas H. Cormen et al. *Introduction to algorithms*. Vol. 6. MIT press Cambridge, 2001.

[6] Pasquale De Meo et al. "A novel measure of edge centrality in social networks". In: *Knowledge-based systems* 30 (2012), pp. 136–150. URL: http://www.sciencedirect.com/science/article/pii/S0950705112000160.

[7] Pasquale De Meo et al. "Generalized louvain method for community detection in large networks". In: *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*. IEEE. 2011, pp. 88–93. URL: http://ieeexplore.ieee.org/document/6121636/.

[8] David Easley and Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010.

[9] *Facebook (NIPS) network dataset – KONECT*. Sept. 2016. URL: http://konect.uni-koblenz.de/networks/ego-facebook.

[10] Santo Fortunato. "Community detection in graphs". In: *Physics Reports* 486.3 (2010), pp. 75–174. URL: http://www.sciencedirect.com/science/article/pii/S0370157309002841.

[11] Michelle Girvan and Mark EJ Newman. "Community structure in social and biological networks". In: *Proceedings of the National Academy of Sciences* 99.12 (2002), pp. 7821–7826. URL: http://www.pnas.org/content/99/12/7821.full.

[12] Madhura Kaple, Ketki Kulkarni, and Katerina Potika. "Viral Marketing for Smart Cities: Influencers in Social Network Communities". In: *9th IEEE International Workshop on Big Data Appications in Smart City Development*. 2017.

[13] Brian W Kernighan and Shen Lin. "An efficient heuristic procedure for partitioning graphs". In: *Bell System Technical Journal* 49.2 (1970), pp. 291–307.

[14] Joseph B Kruskal. "On the shortest spanning subtree of a graph and the traveling salesman problem". In: *Proceedings of the American Mathematical society* 7.1 (1956), pp. 48–50. URL: http://www.jstor.org/stable/2033241?seq=1#page_scan_tab_contents.

[15] James MacQueen et al. “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297. URL: http://www-m9.ma.tum.de/foswiki/pub/WS2010/CombOptSem/kMeans.pdf.

[16] Julian McAuley and Jure Leskovec. “Learning to Discover Social Circles in Ego Networks”. In: *Advances in Neural Information Processing Systems*. 2012, pp. 548–556.

[17] Natarajan Meghanathan. “A Greedy Algorithm for Neighborhood Overlap-Based Community Detection”. In: *Algorithms* 9.1 (2016), p. 8. URL: http://www.mdpi.com/1999-4893/9/1/8/htm.

[18] Tamas Nepusz. *python-igraph 0.7.1.post6*. URL: http://pypi.python.org/pypi/python-igraph (visited on 02/10/2017).

[19] Mark Newman. *American College Football*. 2000. URL: http://www-personal.umich.edu/~mejn/netdata/ (visited on 04/25/2017).

[20] Mark EJ Newman. “Fast algorithm for detecting community structure in networks”. In: *Physical Review E* 69.6 (2004), p. 066133. URL: http://journals.aps.org/pre/abstract/10.1103/PhysRevE.69.066133.

[21] Mark EJ Newman and Michelle Girvan. “Finding and evaluating community structure in networks”. In: *Physical Review E* 69.2 (2004), p. 026113. URL: http://journals.aps.org/pre/abstract/10.1103/PhysRevE.69.026113.

[22] Pascal Pons and Matthieu Latapy. “Computing communities in large networks using random walks”. In: *International Symposium on Computer and Information Sciences*. Springer. 2005, pp. 284–293.

[23] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. "Near linear time algorithm to detect community structures in large-scale networks". In: *Physical Review E* 76.3 (2007), p. 036106. URL: https://arxiv.org/pdf/0709.2938.pdf.

[24] *Single-link and complete-link clustering*. URL: https://nlp.stanford.edu/IR-book/html/htmledition/single-link-and-complete-link-clustering-1.html (visited on 02/10/2017).

[25] Minimum Spanning Tree. "Minimum Spanning Tree". In: (2007). URL: http://s3.amazonaws.com/academia.edu.documents/40444608/19MST.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1493193718&Signature=5gBdCouXz1TtCqwqUjRQ08WMR8k%3D&response-content-disposition=inline%3B%20filename%3DMinimum_Spanning_Tree.pdf.

[26] Wikipedia. *Social network analysis — Wikipedia, The Free Encyclopedia*. [Online; accessed 5-May-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Social_network_analysis&oldid=775992950.

[27] Wayne W Zachary. "An information flow model for conflict and fission in small groups". In: *Journal of Anthropological Research* (1977), pp. 452–473.

[28] Reza Zafarani, Mohammad Ali Abbasi, and Huan Liu. *Social Media mining: An Introduction*. Cambridge University Press, 2014.

References